Capítulo 1

Introducción a ciencia de datos

1.1. -

1.2. Extracción de grandes cantidades de información, uso de disco vs. uso de memoria

En muchas situaciones la cantidad de datos necesarios para realizar algún cómputo es demasiado grande para almacenarse en memoria principal (RAM). En este caso se requiere de un tipo distinto de almacenamiento, los archivos generalmente tienen mucha mayor capacidad, debido al bajo costo por byte de los sistemas de almacenamiento secundario.

Otra ventaja de los archivos, es que son persistentes: si se apaga la computadora, se pierden los datos de la memoria principal mientras que la información almacenada en memoria secundaria se manteniene por tiempo indefinido. Sin embargo su desventaja principal es que el tiempo para accesarla es mucho mayor que el de la memoria principal; esta diferencia de tiempos implica que se deben utilizar técnicas diferentes para manipularla de forma eficiente.

1.2.1. Visualización

Una visualización es efectiva cuando puede ser decodificada con precisión por la audiencia que la utiliza: las características clave deben emerger y las relaciones más sutiles deberían ser visibles de forma inmediata. En esta sección revisaremos algunos métodos y principios para generar visualizaciones útiles, basados en el trabajo de William Cleveland.

La visualización de datos combina un par de ideas: (1) entendimiento de los principios guía de las gráficas y (2) entendimiento básico de los paradigmas de gráficas comúnmente utilizadas para los diferentes tipos de datos. Conocerlas permitirá crear una versión abstracta de la gráfica; la habilidad de esbozar una gráfica es muy importante en su construcción. Además, es importante elegir la herramienta de software para generar las visualizaciones de los datos; en nuestro caso usaremos **ggplot2**, desarrollada por Hadley Wickham; *gg* es acrónimo de *The Grammar of Graphics*.

1.2.1.1. Principios de la visualización de datos

Al hablar de *buena visualización de datos*, hay dos caminos principales: la **simplicidad** y el tratamiento **exhaustivo**. La elección no es tan sencilla como sería deseable, pero conociendo

ciertos principios se puede simplificar un poco; al final, el objetivo es tener buenas visualizaciones y mejorar aquellas con las que ya se cuenta.

Una visualización es efectiva cuando el analista logra transmitir a la audiencia la información que le resulta útil. Si la naturaleza de la información es complicada, es posible que se necesite una visualización complicada, sin ambargo es común que se puedan organizar nuestras ideas en términos de principios.

Let the Data Speak

Deja que los datos hablen. Tufte acuñó un término para describir aque elementos que no agregan entendimiento alguno en una visualización: *chartjunk*. Una buena definición de Robert Kosara es: cualquier elemento de una gráfica que no contribuye en clarificar el mensaje deseado. Las mayores violaciones a este principio por lo general se encontraban en los ajustes por default de Excel, aunque poco a poco se han ido corrigiendo. Para cada elemento de una figura debemos preguntarnos si ese elemento sirve para *permitir que los datos hablen*.

Let the Data Speak Clearly

Deja que los datos hablen *claramente*. Una sútil mejora al punto previo, es permitir que los datos expresen ideas clara y rápidamente. Por ejemplo, la relación de dos variables graficadas en un *scatterplot*, puede mejorarse añadiendo una línea de ajuste. En el otro extremo, una gráfica de pastel con muchas divisiones puede ofuscar el mensaje si las diferencias no son notables.

Choose Graphical Elements Judiciously

Elige los elementos gráficos cuidadosamente. Al crear gráficas, se deben tomar varias decisiones: colores, tipo de línea, ejes, etiquetas, textos, etc; y deben realizarse de manera consciente. El color es bien usado cuando sirve para indicar pertenencia a un conjunto; los ejes pueden destacar ciertas observaciones en un rango; líneas suavizadas pueden ilustrar tendencias en datos bivariados.

Help Your Audience

Ayuda a tu audiencia. Siempre que sea posible modifica los ajustes de la grafica para que ayuden a entender los datos. Por ejemplo, ordenar las observaciones por nombre puede ser útil para que el lector encuentre rápido una en particular; pero si se desea encontrar la mínima y máxima, será mucho mejor ordenarlas por valor. También puede resultar de utilidad etiquetar las observaciones más importantes.

Limit Your Scope

Limita tu alcance. Proyectos interesantes tienden a generar gran cantidad de datos y crece la tentación de intentar mostrar toda la historia en una sola gráfica; pero esto puede resultar contraproducente ya que se puede distraer la atención del mensaje central.

Teniendo en cuenta estos principios generales, mostraremos algunos ejemplos de tipos de datos comúnmente usados.

1.2.1.2. Buenas elecciones

Muchas de las tareas de visualización se definen por el número y tipo de variables que se graficarán. La diferencia más importante es entre datos cualitativos y cuantitativos; una variable categórica se codifica como un *factor* en R. En cuanto al número de variables a graficar, existen enfoques relativamente claros cuando se tienen una o dos. Una vez que se pasa este límite existen principios generales, pero la creatividad juega un rol muy importante; pero no hay que olvidar evitar hacer mucho en una sola gráfica.

Ejemplos

Las gráficas de ejemplo se realizan con R y el paquete *ggplot2*. El paquete gráfico base de R es útil para aprender las técnicas básicas, se pueden controlar varios aspectos de la gráfica, pero construir gráficas más elaboradas no es sencillo. En cambio, ggplot es más flexible y fácil de usar.

The Grammar of Graphics desarrollado inicialmente por Wilkinson y descrito posteriormente a detalle por Wickham. La gramática de ggplot provee al usuario flexibilidad y poder para las visualizaciones. Esta gramática consiste de varios componentes que se combinan para crear las visualizaciones:

1. ***data***. Los datos que se mostrarán; debe ser un *data frame* dentro de R para poder desplegarse con ggplot.

2. ***aes***. Acrónimo de *aesthetic mapping* que le indican a ggplot cómo traducir los datos en elementos gráficos; por ejemplo un mapeo estético identifica las variables en los ejes x, y, o un mapeo que indique el color de puntos o líneas.

3. ***geoms***. Acrónimo de objetos geométricos; son elementos que visualmente retratan los datos. Por ejemplo, líneas, segmentos, barras de error o polígonos.

4. ***stats***. Son transformaciones estadísticas usadas para reducir y resumir los datos.

5. ***scales***. Proveen un mapeo entre los datos crudos y la figura; también se utilizan para crear leyendas y ejes especializados.

6. ***coord***. Este componente se usa para cambiar las escalas de los ejes desde las unidades originales a diferentes unidades, por ejemplo logarítmicas.

7. ***facet***. Divide la gráfica en sub-gráficas de acuerdo a una variable categórica e indica la forma en la que se dispondrán.

El proceso de construcción de una visualización como una serie de capas en ggplot es una mejora notable con respecto al método tradicional. Las figuras se construyen creando primero una base que consiste una gráfica rudimentaria para posteriormente agregar más información y datos sobre la base en forma de capas. Cada capa puede contener información específica de la lista anterior. Como las capas heredan la mayoría de los valores de sus atributos de la gráfica inicial, la especificación de cada capa suele ser muy sencilla dado que solo es necesario cambiar algunos elementos. Por lo tanto, el sistema de capas simplifica mucho la construcción de gráficas complejas: se suele realizar capa por capa; viendo el resultado de cada capa en la gráfica simplificando corregir errores de código.

Para comenzar, se cargan las bibliotecas necesarias:

```
library(ggplot2)
library(scales)
library(RColorBrewer)
```

Las primeras dos (*ggplot*2 y *scales*) están directamente relacionadas con graficación. La última *RColorBrewer*, está basado en el trabajo de Brewer sobre *color y percepción*; su uso es muy recomendado.

Usaremos el conjunto de datos *mpg* incluido en *ggplot*; contiene datos de economía de combustible para 38 modelos de autos:

```
str(mpg)
head(mpg,n=10)
```

Contiene 11 columnas:

♢ *manufacturer*: Nombre del fabricante

♢ *model*: Nombre del modelo

♢ *displ*: Desplazamiento del motor (litros)

♢ *year*: Año de fabricación

♢ *cyl*: Número de cilindros

♢ *trans*: Tipo de transmisión

♢ *drv*: f = tracción delantera, r = tracción trasera, $4 = 4 \times 4$

♢ *cty*: Millas por galón en la ciudad

♢ *hwy*: Millas por galón en carretera

♢ *fl*: Tipo de combustible

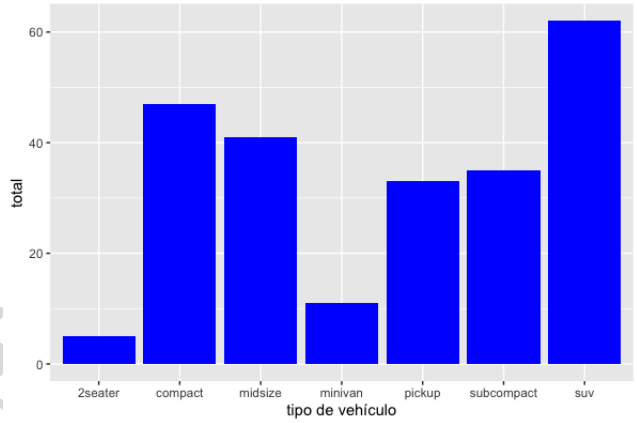
♢ *class*: Tipo de auto

1.2.1.3. Datos univariados

El objetivo principal con datos univariados, típicamente es entender la distribución de los datos. Usualmente comenzamos encontrando el centro, la dispersión y también las observaciones inusuales, conocidos como *valores aislados* (*outliers*). El punto inicial generalmente es obtener el histograma, la función de densidad o alguna de las multiples variantes del diagrama de cajas (*boxplot, violinplot*). También puede resultar útil el diagrama de puntos con barras de error (*dotchart*) cuando el número de categorías no es muy grande.

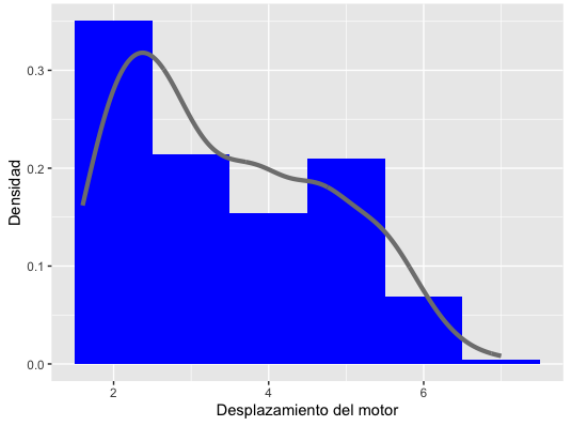
Comenzamos con una gráfica de barras simple:

```
ggplot(mpg) +
  geom_bar(aes(x = class), fill = 'blue') +
  xlab("tipo de vehículo") +
  ylab("total")
```



Para obtener un histograma con su respectiva densidad:

```
ggplot(mpg, aes(x = displ)) +
  geom_histogram(aes(y = ..density..), binwidth = 1, fill='blue') +
  geom_line(stat="density",col="gray50",size=1.5) +
  ylab("Densidad") +
  xlab("Desplazamiento del motor")
```



La *función de densidad empírica* es otra forma de mostrar la distribución de datos cuantitativos; es una estimación de la distribución de probabilidad dirigida por los datos. También puede decirse que es un *histograma suavizado*. Para generarla, se utiliza:

$$\hat{f}_b(x) = \frac{1}{n \times b} \sum_{i=1}^{n} K\left(\frac{x - x_i}{b}\right)$$

donde n es el número de observaciones, b es el ancho del contenedor y K es el *kernel*[1].

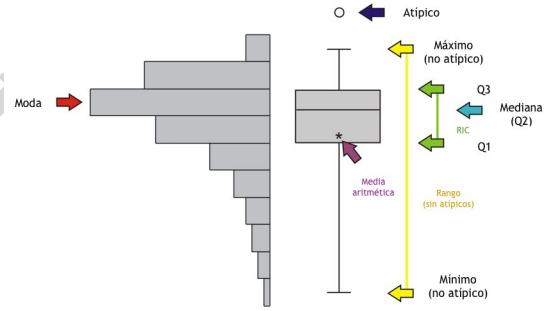
Nota: ggplot incluye variables especiales: *..count..*, *..density..*, etc.; que son resultado de una transformación estadística sobre el conjunto original.

1.2.1.4. Datos bivariados y multivariados

En el caso de datos bivariados, existen tres posibilidades principales: dos variables categóricas, una categórica y otra cuantitativa y dos variables cuantitativas. Para el caso de dos variables categóricas, se puede utilizar una gráfica de mosaico (*mosaicplot*) ya que permite estimar la relación existente entre las categorías.

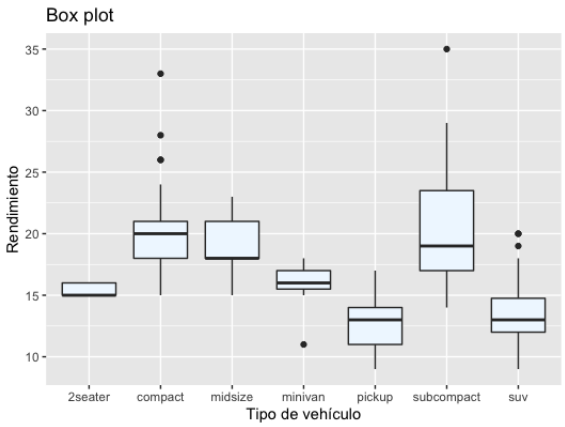
Sin embargo, la mayoría de nuestros datos no son categóricos; para el caso de una variable categórica y una cuantitativa, podemos realizar agrupamientos y obtener medidades de tendencia; en casos simples se realiza automáticamente.

Box plot:



Obteniendo el *box plot* de tipo de auto y rendimiento en la ciudad:

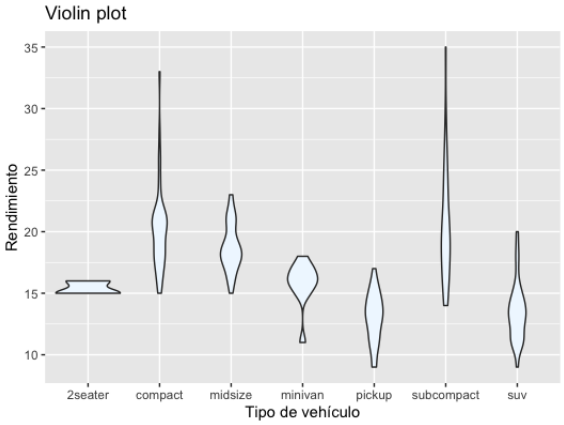
```
ggplot(data = mpg, aes(class, cty)) +
  geom_boxplot(fill='aliceblue') +
  labs(title="Box plot",
       x="Tipo de vehículo", y="Rendimiento")
```



[1]Un *kernel* es una función que se integra a 1 y es simétrica con respecto a 0 y se utiliza para promediar los puntos en el vecindario de un valor dado x. Por default R utiliza un kernel Gaussiano.

La gráfica de *violín* combina un *box plot* con una distribución de densidad:

```
ggplot(data = mpg, aes(class, cty)) +
  geom_violin(fill='aliceblue') +
  labs(title="Violin plot",
       x="Tipo de vehículo", y="Rendimiento")
```



En en casos más complejos, se especifica el tipo de resumen a obtener; para las gráficas con barras de error es necesario tener un resumen de la información en un *dataframe*:

```
library(tidyverse) # permite el uso de 'pipes' %>% para secuencias de acciones
mpg_summary <- mpg %>%
  group_by(class) %>% # agrupado por tipo
  summarize(mean_cty=mean(cty), # media de rendimiento
            sd_cty=sd(cty), # desviación estándar
            N_cty=n(), # total
            se=sd_cty/sqrt(N_cty), # error estándar = sd/N
            x_max=mean_cty+se, # valor máximo
            x_min=mean_cty-se # valor mínimo
  )
```

Reordenamos por la media:

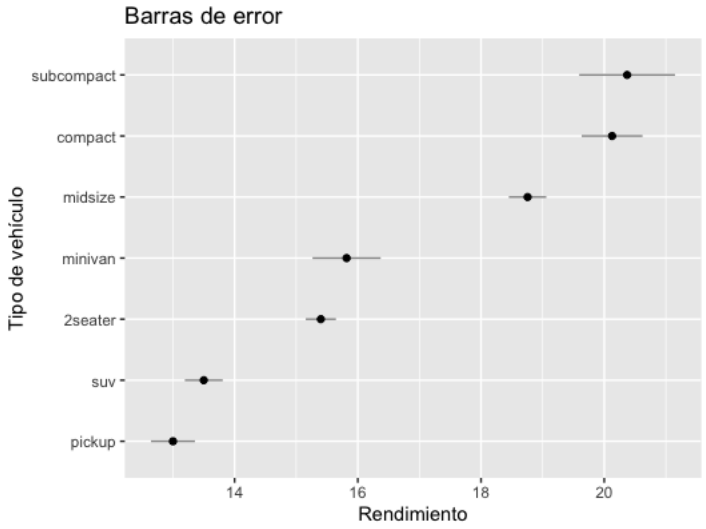
```
mpg_summary$class <- reorder(mpg_summary$class, mpg_summary$mean_cty)
mpg_summary
```

```
  class      mean_cty sd_cty N_cty    se x_max x_min
1 2seater        15.4  0.548     5 0.245  15.6  15.2
```

```
2 compact         20.1  3.39     47 0.494  20.6  19.6
3 midsize         18.8  1.95     41 0.304  19.1  18.5
4 minivan         15.8  1.83     11 0.553  16.4  15.3
5 pickup          13    2.05     33 0.356  13.4  12.6
6 subcompact      20.4  4.60     35 0.778  21.1  19.6
7 suv             13.5  2.42     62 0.307  13.8  13.2
```

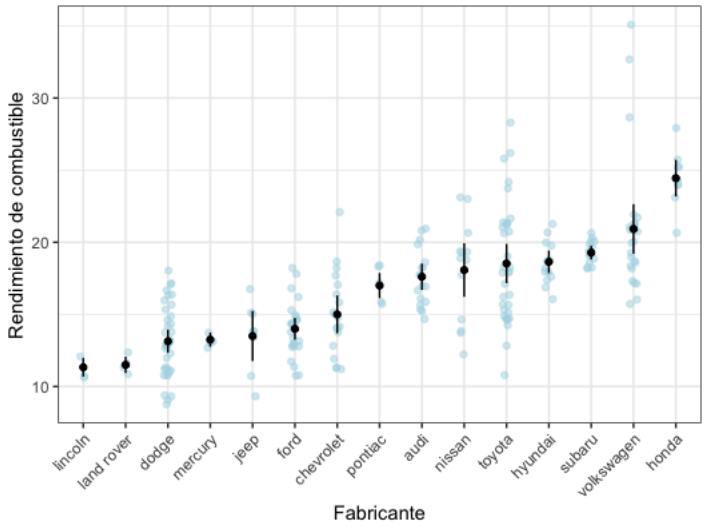
Y podemos ver la gráfica con:

```
ggplot(mpg_summary, aes(mean_cty, y=class)) +
  geom_errorbarh(aes(mean_cty, xmax=x_max, xmin=x_min, y=class),
                 height=0, color="gray60") +
  geom_point(col='black') +
  labs(title="Barras de error",
       x="Rendimiento",y="Tipo de vehículo")
```



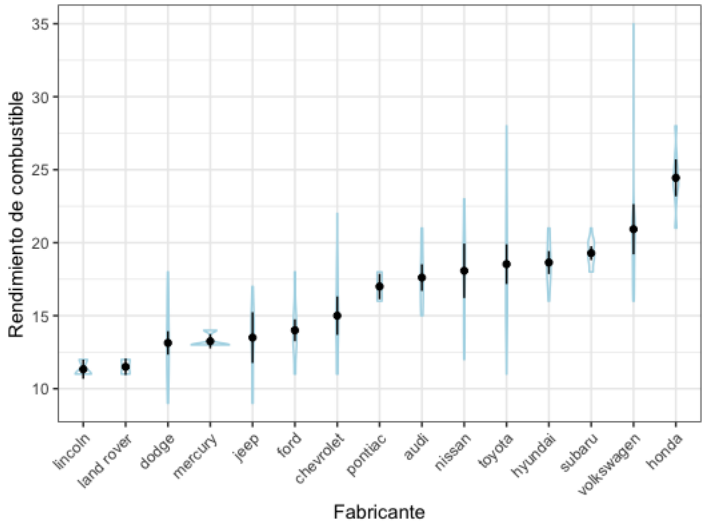
Este ejemplo muestra los datos de rendimiento por fabricante, añadiendo la media y barras de error:

```
ggplot(mpg, aes(x=reorder(manufacturer, cty, FUN=mean), y=cty)) +
  geom_jitter(colour="lightblue", alpha=0.5, width=0.1) +
  geom_point(stat="summary", fun="mean") +
  geom_errorbar(stat="summary", fun.data="mean_se",
                fun.args = list(mult = 1.96), width=0) +
  labs(x="Fabricante", y="Rendimiento de combustible (media + 95%IC)") +
  theme_bw() +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```



Veamos una gráfica similar pero con eometría de violín:

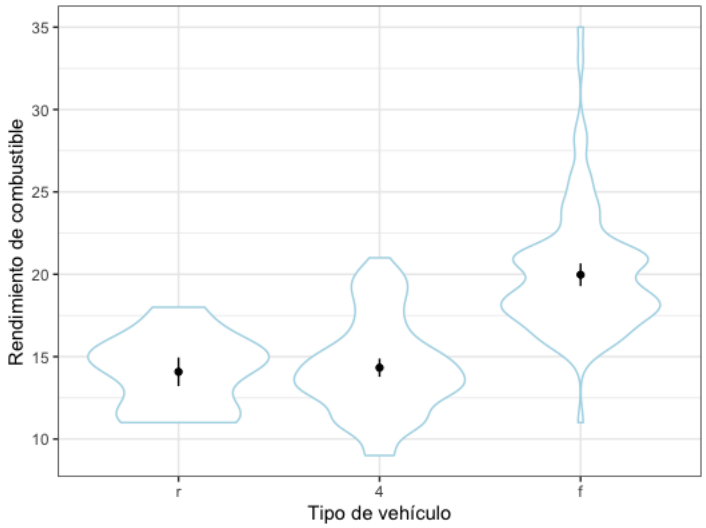
```
ggplot(mpg, aes(x=reorder(manufacturer, cty, FUN=mean), y=cty)) +
  geom_violin(colour="lightblue", alpha=0.5) +
  geom_point(stat="summary", fun="mean") +
  geom_errorbar(stat="summary", fun.data="mean_se",
                fun.args = list(mult = 1.96), width=0) +
  labs(x="Fabricante", y="Rendimiento de combustible") +
  theme_bw() +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```



No funciona muy bien debido a que son muchos fabricantes y cada uno tiene una cantidad relativamente baja de vehículos; luce mejor si en lugar del fabricante usamos el tipo de tracción

de los vehículos:

```
ggplot(mpg, aes(x=reorder(drv, cty, fun=mean), y=cty)) +
  geom_violin(colour="lightblue") +
  geom_point(stat="summary", fun="mean") +
  geom_errorbar(stat="summary", fun.data="mean_se",
                fun.args = list(mult = 1.96), width=0) +
  labs(x="Tipo de vehículo", y="Rendimiento de combustible") +
  theme_bw()
```



Mostramos los datos agrupados por fabricante en un *dotchart* indicando la mediana del rendimiento de combustible en la ciudad.

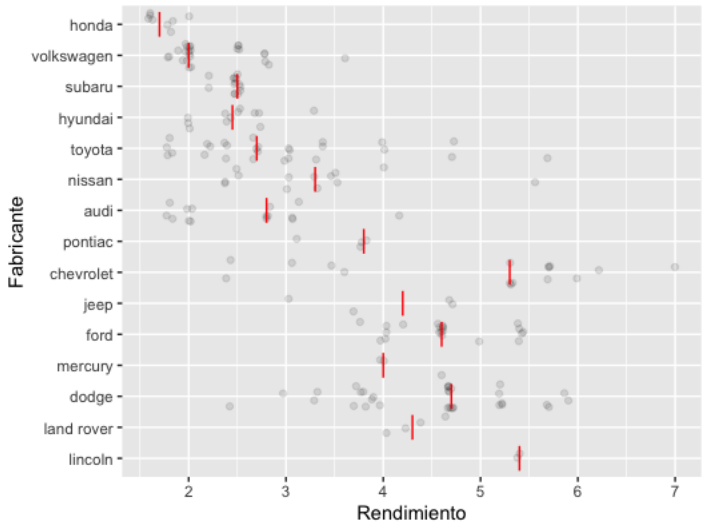
```
# Ejemplo de toma de muestras con la misma semilla para obtener lo mismo
set.seed(3939394)
data <- mpg[sample.int(nrow(mpg),size=200,replace=F),]

# Reordenamiento
data$manufacturer <- reorder(data$manufacturer, data$cty)
```

Obtención de las medianas:

```
medianas <- aggregate(data$displ,
                      list(data$manufacturer),
                      FUN=median)
names(medianas) <- c('manufacturer','displ')
medianas$y_val <- as.numeric(medianas$manufacturer)
# ver el resultado
head(medianas)
```

Graficamos con *geom_points* las observaciones y con *geom_segment* las medias como una línea roja:

```
ggplot(data, aes(x=displ, y=manufacturer)) +
  geom_point(position=position_jitter(h = .4),
             alpha=0.1) +
  labs(x="Desplazamiento",y="Fabricante") +
  geom_segment(data = medianas,
               aes(x = displ, xend = displ,
                   y = y_val -.4, yend = y_val+ .4),
               col="red")
```



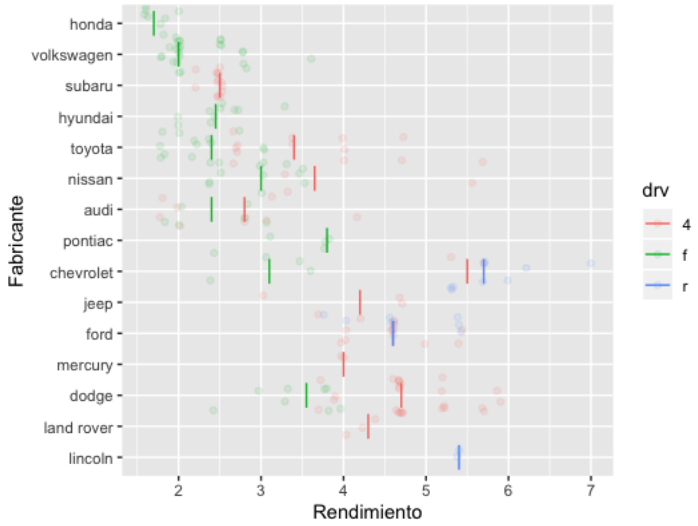
Finalmente, en el caso de dos variables cuantitativas, una buena opción es presentarlas en una gráfica de dispersión (*scatterplot*) añadiendo por ejemplo, líneas y colores para distinguir clases.

Cuando tenemos datos multivariados, comúnmente se presentan como una gráfica de las anteriores añadiendo características a los datos presentados; para el caso de variables cuantitativas (*scatterplot*) se puede añadir una barra de colores que indique el valor de una variables adicionales, también puede cambiarse el tamaño de los puntos.

Para el caso de dos variables categóricas y una numérica, se pueden hacer agrupaciones adicionales: por ejemplo, extendiendo el ejemplo de *Rendimiento-Fabricante*; se pueden ver subgrupos por el tipo de tracción (*drv*):

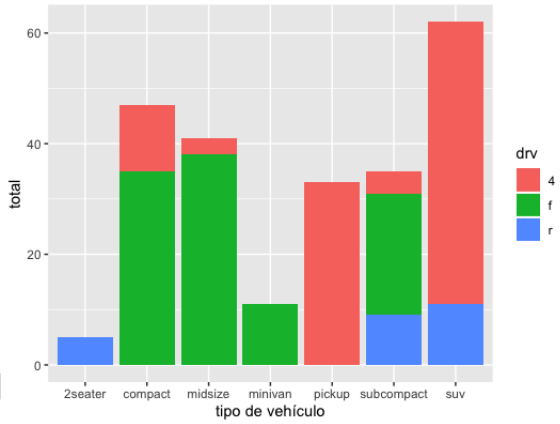
```
medianas <- aggregate(data$displ,
                      list(drv=data$drv,
                           manufacturer=data$manufacturer),
                      FUN=median)
names(medianas) <- c('drv','manufacturer','displ')
medianas$y_val <- as.numeric(medianas$manufacturer)
jit_val <- 0.6
```

```
ggplot(data, aes(x=displ, y=manufacturer, group=drv, col=drv)) +
  geom_point(position=position_jitter(h=jit_val), alpha=0.4) +
  labs(x="Desplazamiento",y="Fabricante") +
  geom_segment(data = medianas,
               aes(x = displ, xend = displ, group=drv, col=drv,
                   y = y_val -.4, yend = y_val+ .4))
```



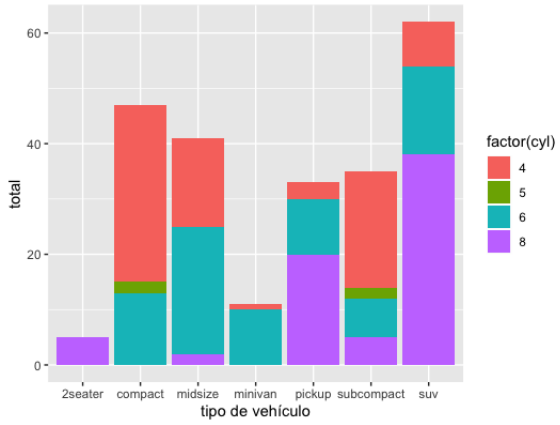
Para obtener una gráfica de barras *apliladas*, se puede usar alguna columna para el llenado de color:

```
ggplot(mpg) +
  geom_bar(aes(x = class, fill = drv)) +
  xlab("tipo de vehículo") +
  ylab("total")
```



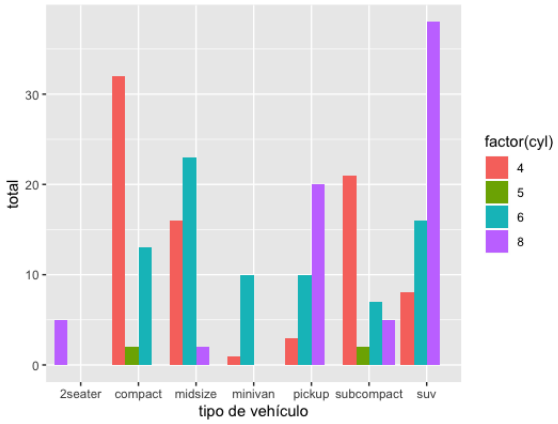
Debe ser un mapeo categórico, para el caso de *cyl*, se debe aplicar *factor*:

```
ggplot(mpg) +
  geom_bar(aes(x = class, fill = factor(cyl))) +
  xlab("tipo de vehículo") +
  ylab("total")
```



También se puede mostrar sin estar apilado:

```
ggplot(mpg) +
  geom_bar(aes(x = class, fill = factor(cyl)),
           position = position_dodge(preserve = 'single')) +
  xlab("tipo de vehículo") +
  ylab("total")
```



Algunos ejemplos en Python (https://bit.ly/39j3kja).

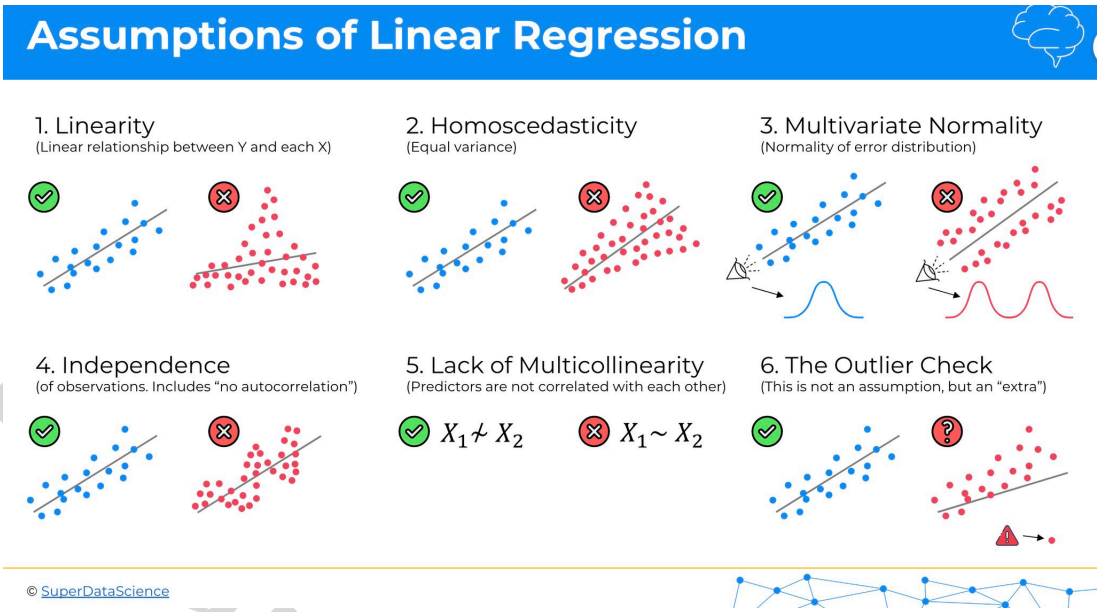
1.2.2. Métodos de regresión lineal

La regresión lineal es un área bien desarrollada y conocida de la estadística; la ubicuidad de sus métodos en el análisis de datos surge de la facilidad con la que se pueden ajustar modelos para describir las características principales de un proceso o una población. Además de ser útil para la descripción, también es muy útil para la predicción ya que los modelos obtenidos en general proveen buenas aproximaciones de relaciones complejas. Analizaremos dichos métodos para entender el potencial de éxito y falla que tienen; sin emabargo, el enfoque apunta a entender los aspectos esenciales y más útiles para la ciencia de datos: **los modelos ajustados**.

Introducción

El contenido de esta sección es aplicable a una amplia variedad de problemas de análisis de datos; estos problemas comparten una característica común: Una de las variables tiene mayor importancia y el objetivo es comprender mejor el proceso que la generó, además de poder predecir valores futuros de dicha variable. Dado que una parte importante de la ciencia de datos incluye análisis predictivos, entender regresión lineal es un componente importante para el análisis de datos. Debe recordarse que la regresión lineal puede resultar muy exitosa tanto para aprender sobre el origen de los datos como para predecir valores futuros y desconocidos de un proceso.

Es importante recordar los supuestos de la regresión lineal:



https://bit.ly/3YcxNe8

1.2.2.1. El modelo de regresión lineal

En el marco de la regresión lineal, los datos pueden verse como n parejas: $D = \{(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)\}$. La visión estadística supone que $y_i, i = 1, 2, \ldots, n$ provienen de una variable aleatoria, identificada como la variable de respuesta Y_i y que el vector explicativo que la acompaña $\mathbf{x}_i$ no es aleatorio y puede usarse para explicar y_i. En ocasiones, el objetivo estadístico es predecir un valor no observado y_0 a partir de un vector observado $\mathbf{x}_0$ usando una función de predicción obtenida a partir de los datos. En estadística se predicen valores de una variable aleatoria y se estiman los parámetros que describen una distribución de la misma.

Las variables $\mathbf{x}_i$, y_i son intrínsecamente diferentes; el objetivo es describir o modelar el valor esperado de la variable Y_i mientras que $\mathbf{x}_i$ es un vector de interés secundario. Comúnmente se asume que el vector explicativo no tiene una distribución, en cambio contiene valores fijos medidos sin error o en control absoluto del investigador. Esta suposición puede resultar engañosa y la regresión lineal es un método valioso para obtener información sobre el proceso o la población de la que provienen los datos. El vector $\mathbf{x}$ se conoce como *explicativo* o *predictor*, dependiendo del objetivo particular del análisis; por simplicidad se utilizará el término *vector predictor* sin importar el tipo de análisis que se esté realizando.

El modelo de regresión lineal especifica que el valor esperado de Y_i es una función lineal de $\mathbf{x}_i$ dado por:

$$E\left(Y_i|x_i\right) = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p}$$

O en su versión vectorial:

$$E\left(Y_i|x_i\right) = \mathbf{x}_i^T \boldsymbol{\beta}$$

donde $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_p]^T$ es un vector de constantes desconocidas. El vector $\mathbf{x}_i = [1, x_{i,1}, \ldots, x_{i,p}]$ contiene las observaciones de los p predictores variables; la longitud de los vectores $\boldsymbol{\beta}$ y $\mathbf{x}_i$ es $q = p + 1$.

La aplicación de la predicción es simple en el sentido de que los detalles del modelo predictivo no tienen especial importancia si produce predicciones precisas de Y; es más común que se tenga interés en entender la influencia de cada variable predictora sobre Y, más precisamente sobre el valor esperado de Y. El modelo de Y_i como función lineal de $\mathbf{x}_i$ puede expandirse para especificar la distribución de Y_i.

Además, si la distribución satisface ciertas condiciones (como normalidad), se pueden realizar un conjunto extendido de inferencias. Estas inferencias adicionales dependen de condiciones llamadas *modelo de inferencia*, el cual establece que:

$$\begin{aligned} Y &= E(Y|\mathbf{x}) + \epsilon \\ E(Y|\mathbf{x}) &= \mathbf{x}^T\boldsymbol{\beta} \\ \epsilon &\sim N\left(0, \sigma_\epsilon^2\right) \end{aligned}$$

La tercera condición especifica que la variable aleatoria *residual* ϵ se distribuye como una normal con esperanza 0 y varianza σ_ϵ^2. La varianza de ϵ es independiente de la varianza de x y es la misma para cada valor de $E(Y|\mathbf{x})$. La varianza constante implica que las diferencias entre los valores reales de Y y las estimaciones de $E(Y|\mathbf{x})$ deberían ser aproximadamente los mismos.

1.2.2.2. Mínimos cuadrados

Explicación en Prometeo: https://bit.ly/3AYiwPN.

La primera tarea computacional del análisis de regresión lineal es calcular un estimado del vector de parámetros $\boldsymbol{\beta}$. En estadística y ciencia de datos, el estimador de mínimos cuadrados casi siempre es la primera elección porque es fácil de entender y calcular. Aún más, su precisión compite con la de una gran variedad de métodos más complejos; los intervalos de confianza y las pruebas de hipótesis con respecto a los parámetros $\beta_0, \beta_1, \ldots, \beta_p$ son sencillos.

El objetivo de los mínimos cuadrados es minimizar la suma de los residuos al cuadrado; los residuos son las diferencias entre el valor observado y_i y los valores ajustados $\hat{y}_i = \mathbf{x}_i^T\hat{\boldsymbol{\beta}}, i = 1, \ldots, n$, donde $\hat{\boldsymbol{\beta}}$ es un vector de números reales de longitud q. La suma de los residuos al cuadrado se calcula con:

$$\begin{aligned} S\left(\hat{\boldsymbol{\beta}}\right) &= \sum_{i=1}^n \left(y_i - \hat{y}_i\right)^2 \\ &= \sum_{i=1}^n \left(y_i - \mathbf{x}_i^T\boldsymbol{\beta}\right)^2 \end{aligned}$$

Se minimiza con la elección de $\boldsymbol{\beta}$; al tener valor desconocido necesitamos calcularlo para poder continuar. El vector $\hat{\boldsymbol{\beta}}$ que minimiza $S(\cdot)$ se conoce como el estimador de mínimos cuadrados y, por definición, cuaquier otro vector entrega una suma de errores al cuadrado al menos tan grande como este estimador. Una derivación puede consultarse en https://bit.ly/2KfhAyl. Lo importante desde nuestra perspectiva es que el estimador $\boldsymbol{\beta}$ es la solución de las *ecuaciones normales*:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

$\underset{n\times q}{\mathbf{X}}$ se forma al *apilar* los vectores predictivos $\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T$. El vector $\mathbf{y} = [y_1, \ldots, y_n]^T$ consiste de n productos de las variables aleatorias $Y_1, \ldots, Y_n$. Si $\mathbf{X}^T\mathbf{X}$ es invertible, entonces la solución de la ecuación normal es:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Si el modelo inferencial anterior describe aproximadamante bien la relación entre $E(Y)$, $\mathbf{x}$ y la distribución de ϵ, entonces la varianza de Y con respecto a su esperanza condicional $E(Y|\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta}$ se estima como:

$$\sigma_\epsilon^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-q}$$

donde $\hat{y} = \mathbf{x}_i^T\hat{\boldsymbol{\beta}}$. La varianza del estimador $\hat{\boldsymbol{\beta}}$ es la matriz de $q \times q$:

$$var\left(\hat{\boldsymbol{\beta}}\right) = \sigma_\epsilon^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$$

La diagonal de los elementos de $var\left(\hat{\boldsymbol{\beta}}\right)$ son las varianzas de los estimadores individuales $\hat{\beta}_0, \ldots, \hat{\beta}_p$, mientras que los elementos fuera de ella son las covarianzas entre los estimadores. El estimador usual de la matriz de varianzas es $\hat{var}\left(\hat{\boldsymbol{\beta}}\right) = \hat{\sigma}_\epsilon^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$. La varianza estimada para un estimador particular $\hat{\beta}_i$ se extrae de la diagonal de $\hat{var}\left(\hat{\boldsymbol{\beta}}\right)$. La raíz cuadrada de la varianza se conoce como el error estándar de $\hat{\beta}_i$:

$$\hat{\sigma}\left(\hat{\boldsymbol{\beta}}_i\right) = \sqrt{\hat{\sigma}^2\left(\hat{\boldsymbol{\beta}}_i\right)}$$

Los errores estándar se utilizan de forma extensiva para construir intervalos de confianza y para pruebas de hipótesis. Hay que recordar que la propiedad de estos estimadores depende de qué tan real es el modelo inferencial.

Las estimaciones *puntuales* $\hat{\beta}_i, i = 0, 1, \ldots, p$, pueden interpretarse como la mejor estimación de β_i; a pesar de ser la mejor, debe tenerse en cuenta que casi con seguridad no será exactamente igual a β_i. En general existirán otros valores además de la estimación puntual cercanos al punto a estimar que son consistentes con los datos, en el sentido que respaldan la posibilidad de que cada uno podría ser el valor de β_i. Comúnmente se desea presentar un intervalo de valores consistentes además de la estimación puntual: un intervalo de confianza.

Ejemplo: Depresión

Ilustraremos una situación en la que el modelo de inferencia lineal es útil con ayuda del conjunto de datos *Ginzberg*. Para el tratamiento de la depresión (enfermedad mental que puede llegar a ser incapacitante), las causas deben ser entendidas; se ha postulado que está asociada a un par de condiciones menos complejas: *fatalismo* y *simplicidad*. El fatalismo se refiere a la imposiblidad de controlar el mundo en el que vive la persona mientras que la medida en la que una persona observa eventos y circunstancias como positivas o negativas se describe con el término simplicidad. El grado en el que estas variables se asocian con la depresión puede iluminar el valor del tratamiento de la depresión al tomar en cuenta las percepciones de la persona. Medir cuantitativamente la relación entre fatalismo y simplicidad con la depresión puede ser un paso muy importante para comprender las relaciones entre estas condiciones.

El conjunto contiene 82 observaciones realizadas a pacientes hospitalizados por depresión; se calificaron en cuanto a la severidad de la depresión, simplicidad y fatalismo, las variables se escalaron de forma que los valores grandes reflejen manifestaciones mas fuertes de simplicidad

y fatalismo, además de una depresión más severa. La intensidad de la asociación lineal entre fatalismo y depresión, medida con el coeficiente de correlación de Pearson[2] es $r = 0.657$ y la correlación entre simplicidad y depresión es $r = 0.643$; estos coeficientes indican un grado de asociación lineal moderadamente positivo entre la depresión y cada variable, además muestra cierta certeza en la existencia de una relación conjunta entre depresión, fatalismo y simplicidad. La tarea es cuantificar la intensidad de la asociación conjunta y describir la relación: producir una aproximación objetiva y manejable de una relación complicada y cambiante a nivel de cada paciente.

Consideremos el modelo lineal:

$$E\left(Y|x\right) = \beta_0 + \beta_1 x_{fatalism} + \beta_2 x_{simplicity} = \mathbf{x}^T\boldsymbol{\beta}$$

donde Y es el nivel de depresión y $\mathbf{x} = [1, x_{fatalism}, x_{simplicity}]^T$. En realidad este modelo es sólo una aproximación a la relación real, teniéndolo en cuenta, se realiza el procesamiento con la esperanza de obtener información útil.

Los parámetros obtenidos de la estimación se observan en la tabla:

Variable	Parámetro estimado	Error estándar	Intervalo de confianza 95 % Límite inferior	Límite superior
Constante	0.2027	0.0947	0.0133	0.3921
Fatalism	0.4178	0.1006	0.2166	0.6190
Simplicity	0.3795	0.1006	0.1783	0.5807

Cuadro 1.1: Estimación de los parámetros, error estándar e intervalos de confianza para el modelo

Podemos interpretar matemáticamente los parámetros estimados; dado que los valores de fatalismo y simplicidad se escalaron para tener una media común de valor 1 y desviación estándar de 0.5, el parámetro estimado se puede comparar directamente. En particular, un incremento de una unidad en fatalismo con simplicidad constante induce un incremento estimado de 0.4178 en la escala de depresión, y se estima que una unidad incrementada en la simplicidad con fatalismo constante incrementa el nivel de depresión en 0.3795 unidades; por tanto, se puede argumentar que fatalismo y simplicidad son casi equivalentes para determinar el nivel de depresión, pero que el fatalismo es más importante. Por supuesto que el contexto es muy artificial: es prácticamente imposible que se pueda incrementar el valor de fatalismo o simplicidad (o cualquier otra actitud, creencia, etc.) manteniendo la otra fija; aún más, fatalismo y simplicidad está correlacionadas ($r = 0.631$). En general, los individuos con valores mayores de fatalismo tendrán también valores mayores en simplicidad. Posteriormente se explicará el significado de los intervalos de confianza mostrados en la tabla.

La figura siguiente muestra los datos y el plano generado por el modelo $\hat{y} = 0.2027 + 0.4178x_{fatalism} + 0.3795x_{simplicity}$.

```
1  install.packages('plot3D')
2  install.packages("car")
3  install.packages("rgl")
```

[2]Mide la relación estadística (asociación) entre dos variables continuas, está basado en el método de la covarianza.

```r
library("car") # Ginzberg dataset
library("rgl")
options(rgl.printRglwidget = TRUE)
plot3d(depression ~ fatalism + simplicity, Ginzberg, xlim= c(0,3), zlim=c(0,3))
n <- 20
x <- y <- seq(0, 3, length = n)
region <- expand.grid(x = x, y = y)
z <- matrix(0.2027+0.4178*region$x + 0.3795*region$y, n, n)
surface3d(x, y, z, back = 'line', front = 'line',
          col = 'red', lwd = 1.5, alpha = 0.4)
```

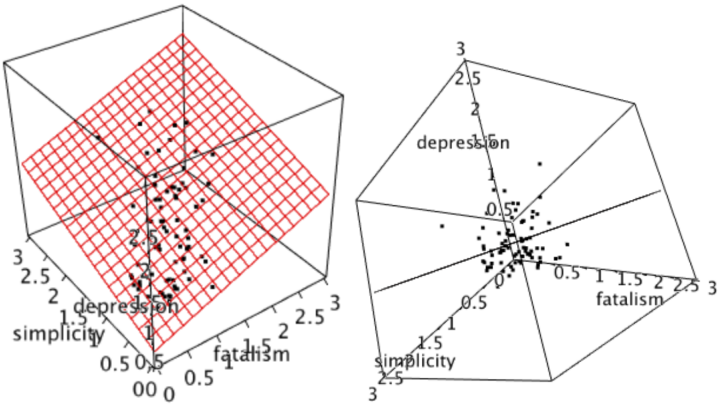


Figura 1.1: Plano del modelo de depresión ajustado usando *rgl*

```r
library("car") # Ginzberg dataset
library("plot3D")
# datos
x <- Ginzberg$fatalism
y <- Ginzberg$simplicity
z <- Ginzberg$depression
# lm para calcular el modelo de regresión lineal ax + by + cz + d = 0
# modelo ajustado (z = ax + by + d)
fit <- lm( z ~ x + y )
# predecir los valores  sobre una malla regular xy
grid_lines = 26
x_pred <- seq(min(x), max(x), length.out = grid_lines)
y_pred <- seq(min(y), max(y), length.out = grid_lines)
xy <- expand.grid( x = x_pred, y = y_pred)
z_pred <- matrix(predict(fit, newdata = xy),
                 nrow = grid_lines, ncol = grid_lines)
# puntos ajustados para las líneas sobre la superficie
fitpoints <- predict(fit)
# scatter con plano de regresión
scatter3D(x, y, z, pch = 18, cex = 2,
          theta = 20, phi = 20, ticktype = "detailed",
          xlab = "fatalism", ylab = "simplicity", zlab = "depression",
          surf = list(x = x_pred, y = y_pred, z = z_pred,
                      facets = NA, fit = fitpoints), main = "Ginzberg")
```

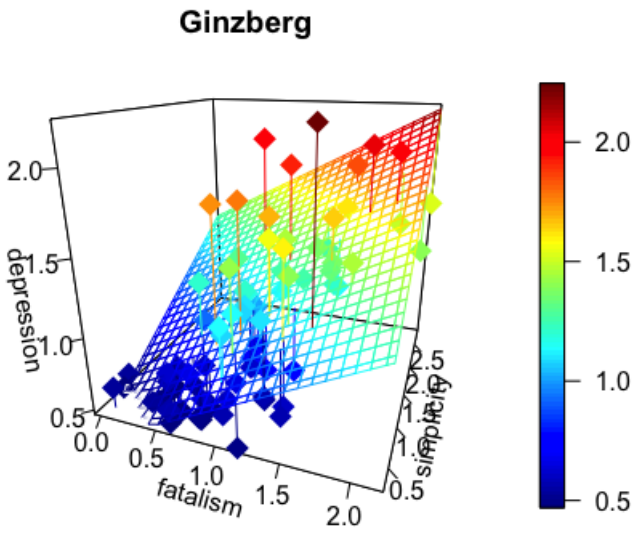


Figura 1.2: Plano del modelo de depresión ajustado usando *plot3D*

La aseveración sobre el cambio en la variable de respuesta (depresión) como una función de las variables explicativas (fatalismo y simplicidad), es un ejemplo de inferencia estadística; esta afirmación está basada en una muestra de datos, pero se puede generalizar a una población mayor de individuos. Pero debe tenerse cuidado, ya que la inferencia podría estar equivocada (incluso bastante), hasta el extremo de *no* existir relación entre las variables.

La posibilidad de una descripción incorrecta nace de que las estimaciones fueron calculadas a partir de una muestra, además de que el modelo propuesto es sólo una aproximación. El nivel del error para cada estimación particular se cuantifica con ayuda del error estándar; la cual puede ser interpretada como la diferencia absoluta promedio entre un parámetro verdadero (desconocido) y una estimación del mismo, determinada con una muestra de observaciones independientes, extraídas de la población o el proceso. La tabla 1.1 incluye los errores estandar $\hat{\sigma}_1\left(\hat{\beta}_1\right)$ y $\hat{\sigma}_2\left(\hat{\beta}_2\right)$ asociados con cada uno de los parámetros; en general, el error estándar no proporciona información útil. En cambio, los intervalos de confianza proveen una interpretación alternativa y muchas veces un mejor enfoque cuando se trata con imprecisiones en estimaciones basadas en muestras.

1.2.2.3. Intervalos de confianza

Un intervalo de confianza para un parámetro es un conjunto de valores posibles para el parámetro que son consistentes con los datos; por ejemplo, la tabla 1.1 presenta un intervalo de confianza de 95 %; para el caso de $\beta_{fatalism}$ se calcula como:

$$\left[\hat{\beta}_{fatalism} - 2\hat{\sigma}\left(\hat{\beta}_{fatalism}\right), \hat{\beta}_{fatalism} + 2\hat{\sigma}\left(\hat{\beta}_{fatalism}\right)\right] = [0.2166, 0.6190]$$

Esto significa que tenemos 95 % de certeza en que el valor de $\beta_{fatalism}$ está entre 0.2166 y 0.6190; todos los valores fuera de este intervalo se consideran como inconsistentes con lo datos y, por tanto, que no confiamos en que el valor verdadero este fuera de este intervalo. El término $2\hat{\sigma}\left(\hat{\beta}\right)$ se conoce como el *error marginal*; el ancho del intervalo es la diferencia entre el límite

mayor y el inferior, es decir, aproximadamente $4\hat{\sigma}\left(\hat{\beta}\right)$ o dos veces el error marginal. Se prefiere que los intervalos de confianza sean estrechos que amplios y si su amplitud es muy pequeña, se dice que es óptimo, dado que al 95 % de confianza el valor verdadero es esencialmente el estimado.

El *nivel de confianza* es la probabilidad de que el procedimiento entregue un intervalo de confianza que incluye al parámetro; matemáticamente el nivel de confianza es $1-\alpha$ con $0 < \alpha < 1$, valores comúnmente usados de α son 0.01, 0.05 y 0.1. El nivel de confianza puede interpretarse de la siguiente forma: si la recolección de la muestra y el cálculo del intervalo de confianza se repitiera muchas veces, entonces $100\,(1-\alpha)$ % de todos los intervalos de confianza contendrán el valor del parámetro (suponiendo que el modelo inferencial es correcto). Debe recordarse que el parámetro se encuentra dentro de un intervalo $[0.2166, 0.6190]$ ó no, pero es imposible determinar cuál de los dos se cumple. Por tanto, no es correcto decir que hay una probabilidad de 0.95 de encontrar a $\beta_{fatalism}$ está entre 0.2166 y 0.6190; la interpretación correcta es que la probabilidad de que el procedimiento genere un intervalo que contenga el valor real del parámetro es $1-\alpha$; por este motivo, sólo se puede establecer que estamos $100\,(1-\alpha)$ % seguro de que el parámetro está entre 0.2166 y 0.6190.

Que el intervalo de confianza contenga al parámetro con probabilidad $1-\alpha$ depende de dos eventos:

1. Que el intervalo esté centrado en el verdadero valor de β, que, a su vez depende de la correctitud del modelo lineal adoptado; si la relación entre $E\,(Y|x)$ y x no es aproximadamente lineal, el intervalo puede no estar centrado en el valor real. La exactitud de una aproximación lineal para una relación no lineal no depende del tamaño de las muestras, es decir, las fallas de un mal modelo no pueden evitarse incrementando el número de observaciones.

2. El error estandar estimado $\hat{\sigma}\left(\hat{\beta}_i\right)$ debe ser una estimación muy precisa de la variabilidad real en cada una de las estimaciones del parámetro en cada muestra y, la precisión de $\hat{\sigma}\left(\hat{\beta}_i\right)$ depende de varias condiciones de la distribución que describe a la población o muestra.

En el tratamiento de los intervalos de confianza hasta este momento se ha supuesto que el tamaño de la muestra n es grande; cuando n es pequeño (menor a 100), se debe realizar un ajuste para corregir la falta de precisión derivada de este tamaño de muestra. Para valores de $n-q < 80$, un intervalo de confianza $100\,(1-\alpha)$ % más preciso para β es:

$$\hat{\beta} \pm t^*_{n-q,\alpha/2}\hat{\sigma}\left(\hat{\beta}\right)$$

donde $t^*_{n-q,\alpha/2}$ es el percentil $100\alpha/2$ de la distribución-$\mathcal{T}$ con $n-q$ grados de libertad. La distribución-$\mathcal{T}$ es muy similar a la distribución normal, pero tiene colas alargadas cuando $n-q < 80$. En otro caso, se puede calcular un intervalo de confianza para β con $\hat{\beta} \pm \hat{\sigma}\left(\hat{\beta}\right)$.

1.2.2.4. Condiciones de la distribución

El nivel de confianza otorgado a un intervalo de confianza sólo es aproximado, dado que su precisión depende del cumplimiento de un conjunto de condiciones que en la realidad no pueden lograrse con total exactitud; si las condiciones son cumplidas *aproximadamente*, entonces el nivel de confianza está cerca de ser verdadero. Comúnmente estas condiciones son conocidas como *suposiciones*, término a veces malinterpretado y que indica que el analista sólo necesita

suponer o *fingir* que dichas condiciones se cumplen para el problema y los datos con los que está tratando. Sin embargo, las condiciones deben ser cumplidas al menos de forma aproximada para obtener resultados confiables. Hay tres condiciones que describen a la distribución de la variable de respuesta y la cuarta describe las observaciones:

1. *Linealidad*: la relación entre la esperanza de Y y x es lineal, en otras palabras, el modelo $E(Y|\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta}$ es correcto.
2. *Varianza constante*: para todo valor de $\mathbf{x}$, la distribución de Y con respecto a $E(Y|\mathbf{x})$ tiene la misma varianza σ_ϵ^2.
3. *Normalidad*: para cada valor de $\mathbf{x}$, la distribución de Y es normal; de forma equivalente, $\epsilon \sim N\left(0, \sigma_\epsilon^2\right)$.
4. *Independencia*: Las observaciones son instancias de n variables aleatorias independientes. Específicamente, si y_i es una instancia de una variable aleatoria Y_i, para $i = 1, \ldots, n$, entonces $Y_1, \ldots, Y_n$ son variables aleatorias independientes.

Las condiciones 1 y 2 especifican la media y la varianza de Y, por lo tanto la condición de distribución normal puede combinarse con las condiciones de media y varianza: $Y \sim N\left(E(Y|\mathbf{x}), \sigma_\epsilon^2\right)$ para todo valor de $\mathbf{x}$.

Combinando todas las condiciones se llega a la condición de que para Y_i, para $i = 1, \ldots, n$, son variables aleatorias distribuídas independientemente como $N\left(\mathbf{x}_i^T\boldsymbol{\beta}, \sigma_\epsilon^2\right)$.

La importancia de cada condición depende del propósito de un modelo ajustado; si no se cumple la condición de linealidad, el modelo estará sesgado localmente y sobre- o sub-estimará $E(Y|\mathbf{x})$ para algunos valores de $\mathbf{x}$. Sin embargo, aún si la relación no es lineal, el *modelo lineal* aún podría proveer una aproximación razonablemente precisa de la relación real desconocida. Si el objetivo es solamente la predicción, entonces la no linealidad de la relación se tolera dado que el modelo ajustado es *suficientemente* preciso para este propósito.

Violaciones a la condición de varianza constante tiene poco efecto sobre los parámetros estimados y, por tanto, en la utilidad predictiva de un modelo ajustado. Varianzas no constantes pueden sesgar los errores estándar $\hat{\sigma}\left(\hat{\beta}_i\right)$; la consecuencia de los errores estándar sesgados es que los intervalos de confianza son o muy anchos o muy delgados y que la probabilidad de que el procedimiento capture el parámetro no sea $1 - \alpha$ como se había propuesto.

La condición de normalidad tiene relativamente poco impacto en la estimación del parámetro y la utilidad predictiva de un modelo ajustado; pero es crítica para los intervalos de confianza si el tamaño de la muestra es pequeño ($n - q < 80$), porque el teorema del límite central asegura que las combinaciones lineales de variables aleatorias será aproximadamente normal para valores grandes de n. Un error común es buscar la normalidad en las respuestas $y_1, \ldots, y_n$; dado que las respuestas tienen esperanza distinta se enturbia el análisis de normalidad. En cambio, la condición de normalidad debe revisarse utilizando los residuos: $\hat{\epsilon}_1 = y_1 - \hat{y}_1, \ldots, \hat{\epsilon}_n = y_n - \hat{y}_n$, o sobre versiones estandarizadas de los residuos. Si el tamaño de la muestra es pequeño y los residuos son claramente no normales, entonces los intervalos de confianza pueden ser inconsistentes con el nivel de confianza.

1.2.2.5. Ejemplo de regresión lineal con R

El conjunto de datos de atletas australianos consiste de mediciones de hematología y morfología recolectadas de atletas de alto rendimiento que participan en 12 deportes. Este conjunto está

disponible en el paquete DAAG de R y puede ser usado como ejemplo del análisis de regresión; el objetivo es determinar la relación entre el pliegue de la piel, el porcentaje de grasa corporal y el género. El porcentaje de grasa corporal se midió sumergiendo a cada individuo em un tanque de agua para determinar la cantidad de agua que desplaza, se cree que es la mejor forma de hacer esta medición. El pliegue cutáneo puede medirse con un *calibre* que puede ayudar a determinar la cantidad de grasa que se encuentra debajo de la piel; generalmente se calcula como el promedio de medidas en diversas partes del cuerpo. Medir pliegues en la piel requiere menos tiempo y es más económico que tener un tanque para determinar la grasa corporal, pero se han presentado preocupaciones respecto a su exactitud para obtener la grasa corporal en niños. Se determinará la relación entre ambas mediciones utilizando datos de atletas australianos con ayuda de R.

Para instalar la biblioteca DAAG:

```
> install.packages('DAAG')
```

Después, en un archivo de *script*, podemos comenzar a escribir:

```
library(DAAG)
head(ais)
str(ais)
```

En la consola de R se puede escribir *?ais* para ver la ayuda de este conjunto y conocer más detalles de sus columnas, especialmente son de interés: *bmi* índice de masa corporal, *ssf* suma de pliegues cutáneos y *pcBfat* porcentaje de grasa corporal.

Para crear una gráfica básica de pligue cutáneo vs porcentaje de grasa, se agrega en el código fuente:

```
plot(ais$ssf,ais$pcBfat)
```

Parece existir una fuerte relación lineal entre ambas variables; se puede agregar una línea de ajuste de mínimos cuadrados con ayuda de las funciones:

- ♢ *lm*: realiza el ajuste de mínimos cuadrados, devuelve un objeto con los elementos de la regresión de y sobre x si el argumento es $y \sim x$.
- ♢ *abline*: extrae la pendiente y la ordenada al origen del objeto que produco *lm* y agrega una línea sobre la gráfica original.

```
abline(lm(ais$pcBfat ~ ais$ssf))
```

En la consola se pueden obtener los atributos del objeto que devuelve la llamada a *lm*:

```
> str(lm(ais$pcBfat ~ ais$ssf))
```

El atributo más importante se llama *coefficients* y se puede almacenar en una variable propia:

```
> b = lm(ais$pcBfat ~ ais$ssf)$coefficients
> b
```

Regresando al *script*, se puede añadir color a los puntos de la gráfica para distinguir hombre de mujeres:

```
hombres = which(ais$sex == 'm')
points(ais$ssf[hombres], ais$pcBfat[hombres], col = 'blue', pch = 16)
points(ais$ssf[-hombres], ais$pcBfat[-hombres], col = 'red', pch = 10)
```

Para ver una resumen estadístico del modelo ajustado, en el *script* se puede añadir:

```
lm.obj = lm(ais$pcBfat~ais$ssf)
summary(lm.obj)
confint(lm.obj)
```

La última línea calcula y muestra el intervalo de confianza del 95 % para β_0 y β_1.

Como la relación es diferente entre hombres que entre mujeres, se puede agregar una variable que ayude a distinguirlos:

```
mujeres = as.integer(ais$sex == 'f')
print(mujeres)
```

Este tipo de variables se conocen como *indicador* y, en este caso se define como:

$$x_{i,mujer} = \begin{cases} 1, & \text{si la i-ésima atleta es mujer} \\ 0, & \text{si el i-ésimo atleta es hombre} \end{cases}$$

Se puede incorporar la variable indicadora en el modelo de porcentaje de masa corporal esperada:

$$E\left(Y|x\right) = \beta_0 + \beta_1 x_{i,ssf} + \beta_2 x_{i,mujer}$$

Ajustando el modelo con esta nueva variable:

```
lm(ais$pcBfat ~ ais$ssf+mujeres)
```

El modelo ajustado resultante para el i-ésimo atleta puede expresarse como:

$$\begin{aligned}\hat{y}_i &= 1.1307 + 0.1579x_{i,ssf} + 2.9844x_{i,mujer} \\ &= \begin{cases} 4.1151 + 0.1579x_{i,ssf}, & \text{si la i-ésima atleta es mujer} \\ 1.1307 + 0.1579x_{i,ssf}, & \text{si el i-ésimo atleta es hombre} \end{cases}\end{aligned}$$

porque $2.9844 + 1.1307 = 4.1151$. Se estima que las atletas australianas tienen 2.98 % más grasa corporal que los hombres con iguales pliegues cutáneos.

Se pueden añadir líneas de tendencia para hombres y mujeres:

```
abline(a=1.1307, b=0.1579, col='blue')
abline(a=4.1151, b=0.1579, col='red')
```

Finalmente, si se desea revisar si hay diferencias en el porcentaje de grasa corporal entre los atletas agrupados por deporte; el uso de una gráfica de tipo *boxplot* puede ayudar:

```
boxplot(ais$pcBfat ~ ais$sport, cex.axis=.8,
        ylab = 'Porcentaje de grasa corporal')
```

El argumento *ces.axis* sirve para reducir el tamaño de las etiquetas mostradas.

Conclusión

El análisis realizado muestra que el uso de medición de pliegues cutáneos puede ser usado para determinar el porcentaje de grasa corporal con poca pérdida de exactitud; sobre todo si se realiza tomando en cuenta el género de los individuos, es decir, separando los análisis de hombres y mujeres.

Programa:

♢ Implementar el cálculo de los estimadores:

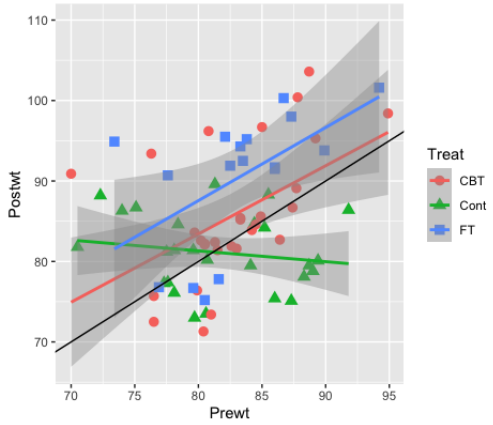
$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

$$\sigma_\epsilon^2 = \frac{\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}{n-q}$$

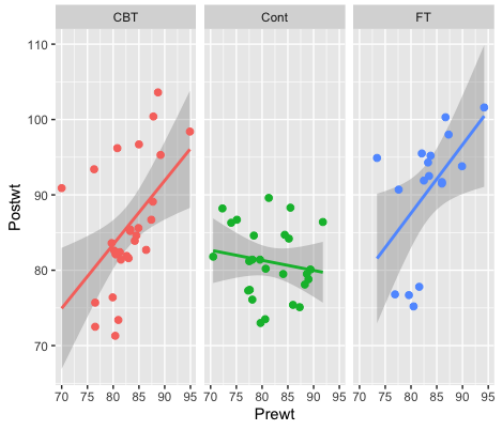
$$var\left(\hat{\boldsymbol{\beta}}\right) = \sigma_\epsilon^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$$

♢ El conjunto de datos *anorexia*, dentro del paquete *MASS* de R reune datos de 72 pacientes que participaron en un experimento para determinar la efectividad de terapias. Se aplicaron tres tipos: terapia familiar (*FT*), terapia cognitiva conductual (*CBT*) y un grupo de control sin terapia (*Cont*). El objetivo del análisis es determinar la efectividad de los tratamientos, así como estimar la media de ganancia de peso, comparando el peso antes y después del tratamiento (*Prewt* y *Postwt*). como evidencia de que son ganancias reales

- Grafica el peso antes y después del experimento identificando los puntos de cada grupo con diferentes formas y colores (como se añadirán más elementos a la gráfica, se sugiere usar *ggplot*)
- Agrega líneas de regresión por mínimos cuadrados usando los mismos colores para identificarlos (ggplot regala los intervalos de confianza)
- Agrega una línea con pendiente 1 y ordenada al origen 0



- ¿Se puede concluir algo sobre la eficacia de los tratamientos?
- Otra forma de ver los datos es dividir la gráfica de acuerdo al tipo de tratamiento; se puede realizar con *facet_grid*:



```
plt = ggplot(anorexia, aes(Prewt, Postwt)) + facet_grid(.~Treat)
plt = plt + geom_smooth(aes(colour = Treat), method ='lm')
plt = plt + geom_point(aes(colour = Treat), size = 2)
plt = plt + scale_colour_discrete(guide="none")
plt
```

- Muestra el resumen de cada modelo de regresión, ¿Para qué grupos existe evidencia de una relación entre los pesos? Explica la diferencia entre el grupo de control y los grupos con terapias; genera una conclusión al respecto.

*

1.2.2.6. Regularización

Un modelo obtenido como la solución de las ecuaciones normales:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y},$$

funciona muy bien si el número de muestras (n) es mucho mayor que el número de parámetros (p):

$$n \gg p \Rightarrow \textit{poca varianza}$$

Cuando esta condición no se cumple, se pueden presentar altas correlaciones entre las variables predictoras, provocando que el modelo esté *sobreajustado*.

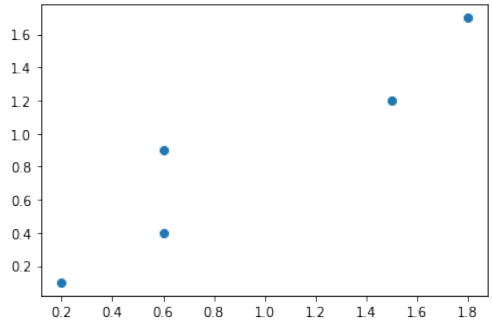
1.2.2.7. Ejemplo de regresión lineal con Python

Revisemos ahora un ejemplo sencillo con Python que servirá para motivar la revisión de un modelo alternativo de regresión lineal.

```
import matplotlib.pyplot as plt
import numpy as np
```

Un conjunto de datos *artificial*:

```
x = np.array([0.2,0.6,0.6,1.5,1.8])
y = np.array([0.1,0.4,0.9,1.2,1.7])
plt.scatter(x,y)
```



Al igual que la *selección* de datos de entrenamiento y prueba:

```
#from sklearn.model_selection import train_test_split
#X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.6)
X_train,y_train = np.array([0.6,1.8]), np.array([0.4,1.7])
X_test,y_test = np.array([0.2,0.6,1.5]), np.array([0.1,0.9,1.2])
```

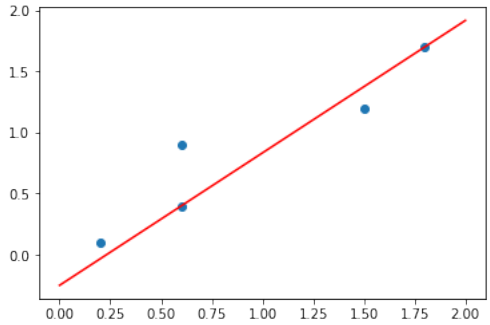
Entrenamos un modelo de regresión lineal, se encuentra dentro de *sklearn.linear_model* y mostramos la recta obtenida:

```
from sklearn.linear_model import LinearRegression

X_train.resize(len(X_train),1)
y_train.resize(len(y_train),1)

modelo = LinearRegression()
modelo.fit(X_train,y_train)

coefs = modelo.coef_[0]
intercept = modelo.intercept_[0]
print("y = {:.4f} + {:.4f}x".format(intercept,coefs[0]))
```

```
y = -0.2500 + 1.0833x
```

Y su gráfica:

```
xx = np.linspace(0.0,2,2)
yy = modelo.predict(xx.reshape(len(xx),1))

plt.scatter(x,y)
plt.plot(xx,yy,c='r')
```



Parece un buen modelo, pero en realidad está sobreajustado, podemos verlo calculando el error cuadrático medio:

```
from sklearn.metrics import mean_squared_error as mse
mse1 = mse(y_test, modelo.predict(X_test.reshape(len(X_test), 1)))
print('Error : {}'.format(mse1))
```

```
Error : 0.09946759259259248
```

Existen alternativas a la regresión lineal convencional que pueden obtener mejores resultados.

Regularización En general, cuando se realiza una regularización, se utiliza el error cuadrático medio (MSE) como función de costo, añadiendo un término que penaliza la complejidad del modelo:

$$J = MSE + \alpha C,$$

C es la medida de complejidad del modelo. Dependiendo de cómo se mide la complejidad, se tienen distintos tipos de regularización. El hiperparámetro α indica qué tan importante es que el modelo sea simple en relación a qué tan importante es su rendimiento. Cuando se utiliza regularización, se reduce la complejidad del modelo al mismo tiempo que se minimiza la función de costo. Existen dos formas básicas: Lasso ($L1$) y Ridge ($L2$).

Regularización *Lasso* ($L1$) En esta regularización, la complejidad C se mide como la media del valor absoluto de los coeficientes del modelo. Esto se puede aplicar a regresión lineal, polinómicas, regresión logística, redes neuronales, máquinas de soporte vectorial, etc. Matemáticamente se define como:

$$C = \frac{1}{n}\sum_{i=1}^{n} |w_1|,$$

con esto, para el caso del MSE, la función de costo queda:

$$J = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha\frac{1}{n}\sum_{i=1}^{n} |w_1|$$

Lasso es útil cuando se sospecha que varios de los atributos de entrada (*características*) son irrelevantes. Al usar Lasso, se fomenta que la solución sea poco densa, es decir, se favorecen que algunos de los coeficientes tengan valor 0. Esto puede ser útil para descubrir cuáles de los atributos de entrada son relevantes y, en general, para obtener un modelo que generalice mejor. Lasso funciona mejor cuando los atributos no están muy correlacionados entre ellos.

Regularización *Ridge* ($L2$) En este tipo de regularización, la complejidad C se mide como la media del cuadrado de los coeficientes del modelo. También se puede aplicar a diversas técnicas de aprendizaje automático. Matemáticamente se define como:

$$C = \frac{1}{2n} \sum_{i=1}^{n} w_1^2,$$

y la función de costo:

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \frac{1}{2n} \sum_{i=1}^{n} w_1^2$$

Ridge funciona bien cuando se sospecha que varios de los atributos de entrada están correlacionados entre ellos. Ridge hace que los coeficientes obtenidos sean más pequeños, esta disminución de los coeficientes minimiza el efecto de la correlación entre los atributos de entrada y hace que el modelo generalice mejor. Ridge funciona mejor cuando la mayoría de los atributos son relevantes.

Regularización *ElasticNet* ($L1$ y $L2$) Combina las regularizaciones $L1$ y $L2$; con el parámetro r podemos indicar que importancia relativa tienen Lasso y Ridge respectivamente. Matemáticamente:

$$C = r \times Lasso + (1 - r) \times Ridge,$$

desarrollado para el caso del error cuadrático medio, se obtiene:

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + r \times \alpha \frac{1}{n} \sum_{i=1}^{n} |w_1| + (1 - r) \times \alpha \frac{1}{2n} \sum_{i=1}^{n} w_1^2$$

ElasticNet se recomienda cuando se cuenta con gran número de características: algunos serán irrelevantes y otros estarán correlacionados.

Nota *scikit-learn* ofrece el hiperparámetro *"penalty"* en muchos de sus modelos; se puede utilizar "$l1$" o "$l2$" en *penalty* para elegir la regularización a usar.

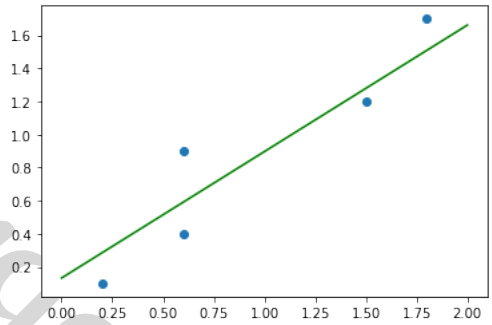
Tomando los mismos valores que el ejemplo de regresión convencional, ahora utilizamos Ridge que también se encuentra dentro de *sklearn.linear_model*:

```
from sklearn.linear_model import Ridge
modelo2 = Ridge(alpha = 0.3)
modelo2.fit(X_train,y_train)
coefs = modelo2.coef_[0]
intercept = modelo2.intercept_[0]
print("y = {:.4f} + {:.4f}x".format(intercept,coefs[0]))
```

```
y = 0.1324 + 0.7647x
```

Y su gráfica:

```
xx = np.linspace(0.0,2,2)
yy = modelo2.predict(xx.reshape(len(xx),1))

plt.scatter(x,y)
plt.plot(xx,yy,c='g')
```



Se observa que la recta de ajuste para este modelo pasa más centrada con respecto a los puntos de la muestra, podemos verlo calculando el error cuadrático medio:

```
from sklearn.metrics import mean_squared_error as mse
mse2 = mse(y_test, modelo2.predict(X_test.reshape(len(X_test), 1)))
print('Error : {}'.format(mse2))
```

```
Error : 0.04533737024221454
```

Este error es menor que el obtenido con el modelo de regresión convencional.

Tarea:

♢ Implementar el cálculo del estimador:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X} - \alpha\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

♢ Iterar el modelo de regresión Ridge variando el valor de α para obtener el mejor para el ejemplo con datos *artificiales* (de preferencia con la implementación propia).

♢ Revisar el uso de las regresiones *Lasso* y *ElasticNet*. Comparar con Ridge.

*

	Model	Penalty function
a)	Ridge	$\|J\|_2^2$
b)	LORETA	$\|\mathbb{L}J\|_2^2$
c)	LASSO	$\|J\|_1$
d)	LASSO Fusion	$\|\mathbb{L}J\|_1$
e)	MPLS	$\sum_k \lambda_k \psi_k(J)$
f)	Fused LASSO	$\lambda_1 \sum_{i=1}^{S-1} \|J_{i,.}\|_1 + \lambda_2 \sum_{i=1}^{S-1} \|J_{i+1,.} - J_{i,.}\|_1$
g)	ENET	$\lambda_1 \|J\|_2 + \lambda_2 \|J\|_1$
h)	ENETL	$\lambda_1 \|\mathbb{L}J\|_2 + \lambda_2 \|\mathbb{L}J\|_1$
i)	Smooth LASSO	$\lambda_1 \|\mathbb{L}J\|_2 + \lambda_2 \|J\|_1$
j)	MXN	$\|J\|_{p,q} = \left(\sum_{t=1}^{T} \left(\sum_{i=1}^{S} \|J_{i,t}\|^p\right)^{\frac{q}{p}}\right)^{\frac{1}{q}}$
k)	GLASSO	$\sum_{s=1}^{S} \left(\sum_{t=1}^{T} \|J_{i,t}\|^2\right)^{\frac{1}{2}}$
l)	ELASSO	$\sum_{t=1}^{T} \left(\sum_{i=1}^{S} \|J_{i,t}\|\right)^2$

Figura 1.3: Otras funciones de penalización (https://bit.ly/3Pxv1dr)

1.2.2.8. Regresión polinomial

En ocasiones una linea recta no es la mejor forma de hacer regresión sobre un conjunto de datos; es común que la relación entre las características y la variable de respuesta (objetivo) no se puede describir correctamente con una línea recta. Es posible que un polinomio funcione mejor para realizar las predicciones de la variable buscada.

Un polinomio puede expresarse de la siguiente manera:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \ldots + \beta_n x^n$$

Donde:

- ♢ y es la variable de respuesta que se desea predecir.
- ♢ x es la característica utilizada para realizar la predicción.
- ♢ β_0 es el término independiente.
- ♢ n es el grado del polinomio.
- ♢ $\beta_1 \ldots, \beta_n$ son los coeficientes que deseamos estimar al ajustar el modelo con los valores de x, y disponibles.

Si comparamos la expresión de la regresión polinomial con la usada para regresión lineal:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_n x_n$$

Notación que son muy similares, esto no es una coincidencia: **la regresión polinomial es un modelo lineal** usado para describir relaciones no lineales; la *magia* recae en crear nuevas características elevando las características originales a alguna potencia.

Por ejemplo, si tenemos una característica x y deseamos un polinomio de tercer grado, la regresión incluirá tanto a x^2 como a x^3 para la estimación de los coeficientes.

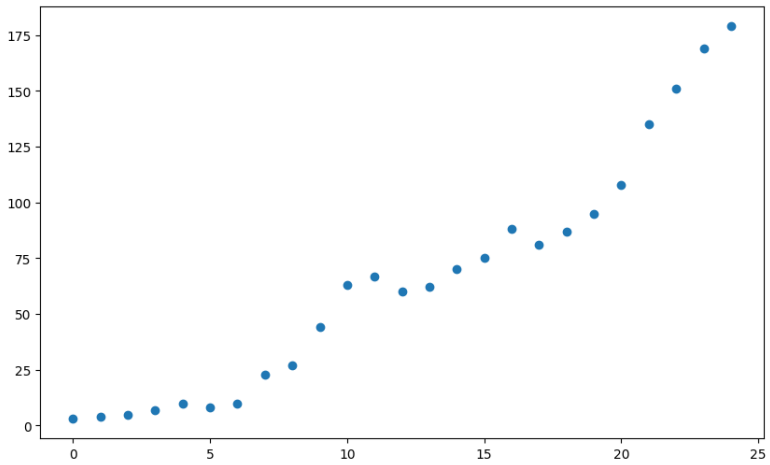
Ejemplo.

Bibliotecas:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Conjunto de datos:

```
y = [3, 4, 5, 7, 10, 8, 10, 23, 27, 44, 63, 67, 60, 62, 70, 75, 88, 81,
     87, 95, 108, 135, 151, 169, 179]
x = np.arange(len(y))
plt.figure(figsize=(10,6))
plt.scatter(x, y)
plt.show()
```



Vemos que no parece funcionar bien una regresión lineal y buscamos una curva que describa mejor la relación. En particular, opdemos inferir que un polinomio de grado 2. Entonces nuestro modelo es:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

Para crear las características polinomiales, existe *PolynomialFeatures* dentro de *sklearn*:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False)
```

include_ bias debe tener valos *false* porque la regresión lineal se encargará del término independiente.

Ajustamos y transformamos el objeto *poly*:

```
poly_features = poly.fit_transform(x.reshape(-1, 1))
poly_features
```

reshape$(-1, 1)$ trnasforma el arreglo de 1D a 2D, necesario para el ajuste. Al observar la salida vemos que el segundo elemento son las características originales elevadas al cuadrado, tal como lo deseamos.

Creamos el modelo lineal:

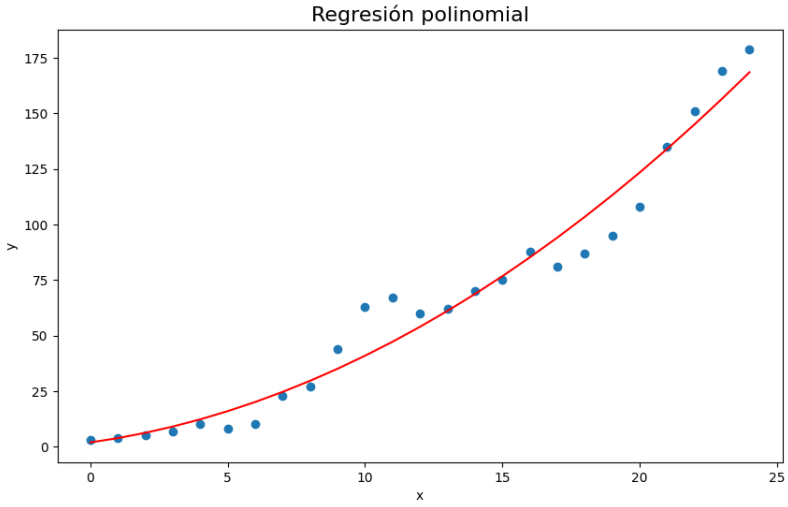
```
from sklearn.linear_model import LinearRegression
poly_reg_model = LinearRegression()
```

Ajustamos y predecimos los valores:

```
poly_reg_model.fit(poly_features, y)
y_pred = poly_reg_model.predict(poly_features)
```

Graficamos el resultado:

```
plt.figure(figsize=(10, 6))
plt.scatter(x, y)
plt.plot(x, y_pred, c="red")
plt.title('Regresión polinomial', size=16)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



1.2.3. Análisis de agrupamiento (*clustering*)

En ocasiones es posible dividir una colección de observaciones en distintos subgrupos, basados solamente en los atributos de las observaciones; cuando se puede hacer esta separación, se facilita el entendimiento de la población o el proceso que generó las observaciones. La intención es realizar división de datos en *grupos* (*clusters*) de observaciones que son más similares dentro de un grupo que entre varios grupos. Los grupos son formados ya sea agregando observaciones o dividiendo un gran grupo de observaciones en una colección de grupos más pequeños.

El proceso de generación de grupos incluye dos variedades de algoritmos:

1. Combinar observaciones entre un número fijo de grupos buscando maximizar la similitud dentro de cada grupo

2. Comenzar con grupos de un solo elemento y, recursivamente combinarlos; alternativamente, se puede comenzar con un sólo grupo y recursivamente dividir nuevos grupos

En esta sección se revisarán dos algoritmos populares y representativos de cada una de estas propuestas: agrupación jerárquica y k-medias. Sea $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ el conjunto de datos, con $\mathbf{x}_i$ un vector de atributos medidos en una unidad observacional.

1.2.3.1. Agrupamiento por $k-$medias

A diferencia del agrupamiento jerárquico por aglomeración, en este algoritmo, el analista establece el número de grupos; el algoritmo comienza asignando arbitrariamente los vectores del conjunto de datos $D = \{x_1, \ldots, x_n\}$ a k grupos; estos grupos iniciales se denotan como $\{A_1, \ldots, A_k\}$. Después, se calculan los *centroides* de cada grupo; estos centroides son medias multivariadas de las observaciones de cada grupo y cada centroide representa al grupo para el cálculo de las distancias. También es posible asignar arbitrariamente k medias iniciales y con ellas generar los grupos.

Es muy poco probable que la configuración inicial sea una buena solución, por tanto el algoritmo itera entre dos pasos:

♢ asignar cada observación al grupo más cercano

♢ recalcular los centroides de cada grupo

Si al menos una observación es reasignada en otro grupo, los centroides cambiarán y debe realizarse otra iteración; el algoritmo continuará pasando entre estos dos pasos hasta que ya no hay reasignaciones. En ese momento, cada observación pertenece al grupo que le es más cercano. A partir de la configuración aleatoria inicial, la suma de cuadrados dentro de cada grupo ha sido minimizado; esto se debe a que dicha suma de cuadrados es equivalente a la suma de distancias euclideanas entre las observaciones y los centroides; además, mover cualquiera de las obervaciones incrementará la suma de distancias (y la suma de cuadrados dentro de los grupos). Por tanto, el algoritmo ha alcanzado *una mejor* asignación posible para las observaciones en los grupos y *un mejor* cálculo de los centroides.

Este algoritmo tiene un atractivo considerable porque minimza una función objetivo popular: la suma de cuadrados. Su inconveniente es que debe generar una configuración inicial aleatoria: una configuración diferente por lo general regresa un resultado distinto.

El centroide del grupo A_i es la media multivariada:

$$\overline{\mathbf{x}}_i = n_i^{-1} \sum_{\mathbf{x}_j \in A_i} \mathbf{x}_j,$$

donde, n_i es el número de observaciones que pertenecen a A_i y $\mathbf{x}_j = [x_{j,1}, \ldots, x_{j,h}]^T$; el número de atributos es h y el $l-$ésimo elemento de $\overline{\mathbf{x}}_i$ es:

$$\overline{x}_{i,l} = n_i^{-1} \sum_{\mathbf{x}_j \in A_i} x_{j,l},$$

para $l = 1, \ldots, h$; la distancia entre cada observación $\mathbf{x}_j$ y un grupo A_i, se define como la distancia entre la observación y el centroide del grupo. Como $\overline{\mathbf{x}}_i = [\overline{x}_{i,1}, \ldots, \overline{x}_{i,h}]^T$, la distancia euclideana cuadrada es:

$$d_E^2 (\mathbf{x}_j, \overline{\mathbf{x}}_i) = \sum_{l=1}^{h} (x_{j,l} - \overline{x}_{i,l})^2.$$

Se puede utilizar la distancia cuadrada en lugar de la distancia, dado que el ordenamiento de más cercano a más distante será el mismo.

Ejemplo

Clasificar las muestras siguientes utilizando $k = 2$:
$[8, 10], [3, 10.5], [7, 13.5], [5, 18], [5, 13], [6, 9], [9, 11], [3, 18], [8.5, 12], [8, 16]$

Ejemplo con Python

Importando bibliotecas:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
```

El conjunto de datos *xclara* es muy usado para probar este tipo de algoritmos:

```
data = pd.read_csv('https://bit.ly/2mi1lqG')
print(data.shape)
data.head()
```

Mostrando la gráfica:

```
f1 = data['V1'].values
f2 = data['V2'].values
X = np.array(list(zip(f1, f2)))
plt.scatter(f1, f2, c='black', s=7)
```

Usando el modelo *KMeans* de *scikit-learn*:

```
from sklearn.cluster import KMeans
# número de medias
kmeans = KMeans(n_clusters=3)
# ajuste
kmeans = kmeans.fit(X)
# etiquetas de cada clase
y = kmeans.predict(X)
# Centroides
C = kmeans.cluster_centers_ y = kmeans.labels_
```

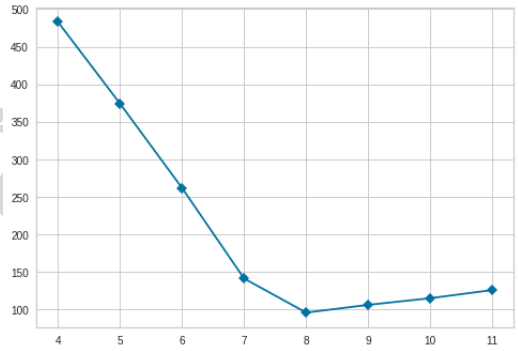
Graficando los resultados:

```
fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], c=y)
ax.scatter(C[:, 0], C[:, 1], marker='*', s=200, c='k')
```

1.2.3.1.1. Determinar un buen valor para *k* Al igual que en modelos anteriores, determinar los hiperparámetros requiere iterar el mismo y encontrar un indicador que nos permita elegir su valor.

Para el caso de $k-$medias, la *gráfica de codo* brinda una buena guía para este fin.

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from yellowbrick.cluster import KElbowVisualizer

X, y = make_blobs(n_samples=1000, n_features=12, centers=8, random_state=42)

visualizer = KElbowVisualizer(KMeans(), k=(4,12), timings=False)
visualizer.fit(X)
plt.xlabel('Número de grupos')
plt.ylabel('Distorsión')
plt.show()
```



Este método recomienda usar como valor para k aquel que corresponde con el codo, es decir, en el que se tiene el mayor cambio en la pendiente.

Clustering Methods: A History of k-Means Algorithms (https://bit.ly/3zQWJ0U)

Ejemplo en 3D

Importando bibliotecas:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
plt.rcParams['figure.figsize'] = (16, 9)
```

Se pueden generar conjuntos de prueba con *make_ blobs*:

```
X, y = make_blobs(n_samples=800, n_features=3, centers=4)
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X[:, 0], X[:, 1], X[:, 2])
```

```
# Objeto KMeans con 4 grupos
kmeans = KMeans(n_clusters=4)
# Ajuste
kmeans = kmeans.fit(X)
# Predicción
labels = kmeans.predict(X)
# Centroides
C = kmeans.cluster_centers_
```

Graficando:

```
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y)
ax.scatter(C[:, 0], C[:, 1], C[:, 2], marker='*', c='k', s=1000)
```

Tarea:

♢ Clasificar las muestras siguientes utilizando k=*2*:
$[1, 12.5], [3, 10.5], [3, 12.5], [3, 14.5], [3, 18], [5, 18], [5, 16], [5, 14.5], [5, 13], [6, 9], [8, 10], [9, 11], [8.5, 12], [7, 13.5], [8, 16], [0.5, 10.5]$

*

1.2.3.2. $k-$*modas*

Es un algoritmo de agrupamiento con una idea similar al aplicado en que se utiliza en $k-$medias, pero se aplica a conjuntos de datos que son mayoritariamente categóricos (como los usados en encuestas).

Sustuye al uso de $k-$medias porque este último algoritmo calcula distancias sobre datos *continuos;* para datos categóricos no es posible obtener esa métrica de distacia. En cambio, $k-$*modas* utiliza las diferencias (errores totales) entre las muestras del conjunto; entre menos diferencias existan los puntos de cada grupo serán más similares.

Al igual que en el caso de $k-$medias, el algoritmo comienza con la elección de k observaciones iniciales como representantes de cada grupo y posteriormente itera entre estos dos pasos mientras no haya convergencia:

♢ Asignar cada observación al grupo más *cercano*, es decir, aquella moda con la que tenga menos diferencias

♢ Recalcular los modas de cada grupo

Ejemplo

Clasificar las muestras siguientes utilizando $k = 2$:
$['x','y','z'], ['y','z','x'], ['z','x','x'], ['y','z','z'], ['x','z','y'], ['z','y','x'], ['x','x','y'], ['z','y','x']$

Ejemplo con Python

Importando bibliotecas:

```
!pip install kmodes
```

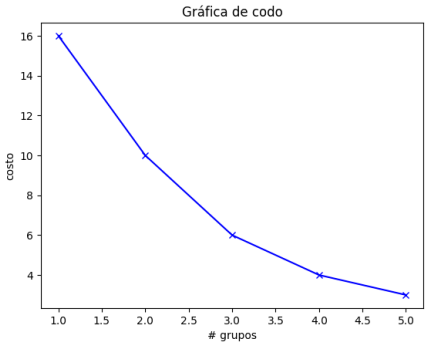
1.2.3.2.1. Determinar un buen valor para *k* Al igual que en modelos anteriores, determinar los hiperparámetros requiere iterar el mismo y encontrar un indicador que nos permita elegir su valor.

Para el caso de $k-$modas, también la *gráfica de codo* brinda una buena guía para este fin.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from kmodes.kmodes import KModes
```

```
col_cabello = np.array(['rubio', 'castaño', 'pelirrojo', 'negro', 'castaño', '
col_ojos = np.array(['azul', 'gris', 'verde', 'café', 'azul', 'gris', 'azul',
tipo_cabello = np.array(['lacio', 'chino', 'ondulado', 'ondulado', 'chino', 'c
personas = ['P1','P2','P3','P4','P5','P6','P7','P8']
data = pd.DataFrame({'person':personas, 'col_cabello':col_cabello, 'col_ojos':
data = data.set_index('person') data
```

```
cost = []
K = range(1,5)
for num_clusters in list(K):
  kmode = KModes(n_clusters=num_clusters,init="random",n_init=5,verbose=False)
  kmode.fit_predict(data)
  cost.append(kmode.cost_)

plt.plot(K, cost, 'bx-')
plt.xlabel('# grupos')
plt.ylabel('costo')
plt.title(Gráfica de codo')
plt.show()
```



Tarea:

♢ Agrega al conjunto de datos una columna continua (como estatura y/o peso) e ingresa los valores correspondientes. Después genera rangos para obtener categorías y obtén los nuevos grupos generados con esa información adicional.

*

1.2.3.3. Agrupamiento jerárquico por aglomeración

Este enfoque comienza con cada observación definiendo un grupo de un sólo elemento: $\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_n\}$, donde n es el número de observaciones. El algoritmo reduce iterativamente el conjunto de grupos combinando grupos similares. La combinación de los conjuntos A y B en alguna iteración se representa como:

$$(A, B) \longrightarrow A \cup B$$

La elección de conjuntos a combinar se puede determinar calculando la distancia entre los grupos y combinando la pareja con la menos distancia intergrupal (más adelante se presentan otras opciones). Por tanto, es requiere una métrica para obtener la distancia entre grupos; por ejemplo, la distancia entre los grupos A y B puede definirse como la distancia más corta entre cualquier vector de A y cualquier vector de B. Matemáticamente:

$$d(A,B) = min\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\},$$

donde d_v es una distancia entre vectores. La distancia euclideana es muy popular para este cálculo. También se puede utilizar la distancia Manhattan (*city-block*), la distancia *city-block* entre $\mathbf{x}_0$ y cada $\mathbf{x}_i \in X$ se define como:

$$d_C(\mathbf{x}_i, \mathbf{x}_0) = \sum_{j=1}^{p} |x_{i,j} - x_{0,j}|,$$

donde p es el número de elementos de los vectores. Se pueden obtener grupos más compactos utilizando la métrica basada en centroides, éste se obtiene para cada grupo como:

$$\overline{\mathbf{x}}_A = n_A^{-1} \sum_{\mathbf{x}_k \in A} x_k,$$

donde n_A es el número de observaciones en el grupo A; entonces la distancia entre A y B es:

$$d_{cent}(A,B) = d(\overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$$

Como se describió anteriormente, el algoritmo combinará grupos hasta obtener uno sólo; pero es necesario revisar conjuntos de grupos intermedios para identificar el que resulte más útil según sea el caso. Para poder identificar el agrupamiento óptimo, se puede utilizar un *dendrograma*, el cual muestra el historial de los agrupamientos, se realiza lo siguiente:

1. Determinar la mayor distancia vertical para la que no hay intersección con otros grupos
2. Trazar una línea horizontal entre ambos extremos
3. El agrupamiento óptimo es igual al número de líneas verticales cruzadas por el trazo anterior

Para el ejemplo de la figura 1.4 la mejor elección sería 4.

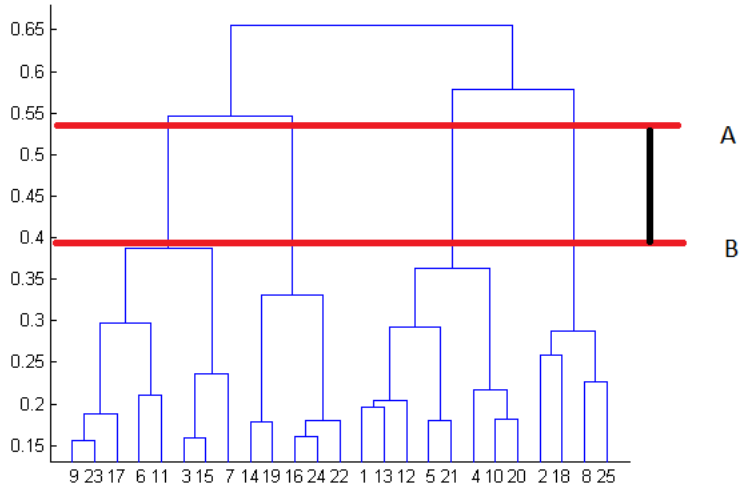


Figura 1.4: Ejemplo de uso de dendrogramas

Criterio de enlazamiento de grupos

Existen otros tipos de enlazamiento entre los grupos, cada uno entrega soluciones distintas.

♢ ***Single linkage***: utiliza la distancia más corta entre dos vectores de cada grupo
$L(A,B) = min\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\}$

♢ ***Complete linkage***: toma en cuenta la distancia más grande entre dos vectores de cada grupo
$L(A,B) = max\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\}$

♢ ***Average linkage***: usa la distancia promedio entre todos los vectores de cada grupo

♢ $L(A,B) = \frac{1}{n_A n_B} \sum_{k=1}^{n_A} \sum_{l=1}^{n_B} d_v(\mathbf{x}_k, \mathbf{x}_l), \mathbf{x}_k \in A, \mathbf{x}_l \in B$

♢ ***Ward linkage***: la distancia es la suma de las diferencias al cuadrado en los grupos

♢ $d_v(A,B) = (\overline{\mathbf{x}}_A - \mathbf{x}_k)^2 + (\overline{\mathbf{x}}_B - \mathbf{x}_l)^2, \mathbf{x}_k \in A, \mathbf{x}_l \in B$

Ejemplo

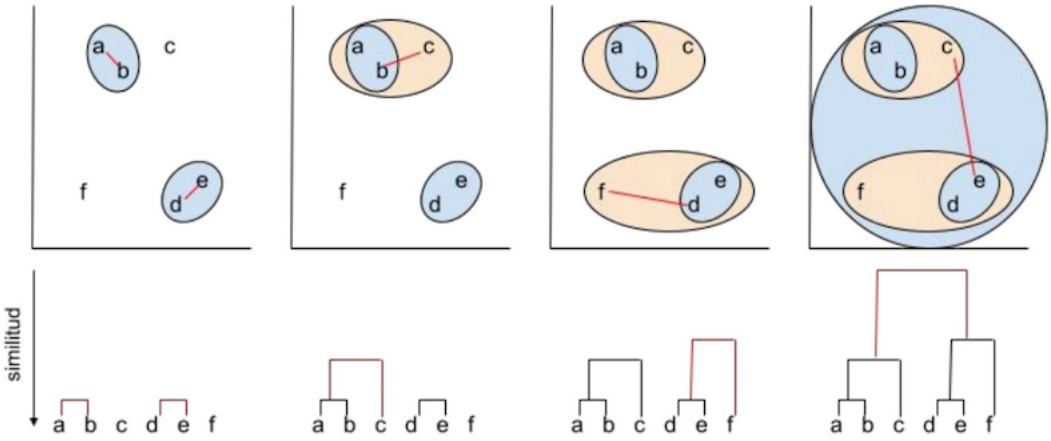


Figura 1.5: Ejemplo de aplicación de *single linkage*

Ejemplo con Python

Importar bibliotecas:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Leer en conjunto de datos:

```
#https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20c
url = 'https://bit.ly/2COHM14'
data = pd.read_csv(url)
data.head()
```

Este conjunto tiene diferentes categorías de productos (*Fresh, Milk, Grocery,* etc.); los valores representan el número de unidades compradas por cada cliente. El objetivo es agrupar a los clientes con compras similares a partir de los datos.

Antes de aplicar el algoritmo, se normalzan los datos para que todas las variables estén a la misma escala; esto es necesario para que el modelo no sea sesgado por las variables con magnitudes mayores:

```
from sklearn.preprocessing import normalize
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
data_scaled.head()
```

Antes de realizar los agrupamientos, es recomendable realizar el dendrograma para determinar el número óptimo de grupos:

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograma")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
```

El eje x contiene las muestras mientras el y representa las distancias entre las mismas. La línea vertical con mayor distancia es la azul, por tanto, se establece como umbral el valor 6 y se traza una línea que cruce al dendrograma:

```
plt.figure(figsize=(10, 7))
plt.title("Dendrograma")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
plt.axhline(y=6, color='r', linestyle='--')
```

Como la línea cruza el dendrograma en dos puntos, obtenemos dos grupos.

Ahora se aplica el algoritmo de agrupamiento jerárquico por aglomeración

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean',
                                  linkage='ward')
cluster.fit_predict(data_scaled)
```

Se observan sólo valores de 0 para el primer grupo y 1 para el segundo, dado que elegimos dos grupos.

Se pueden visualizar los grupos:

```
plt.figure(figsize=(10, 7))
plt.scatter(data_scaled['Milk'], data_scaled['Grocery'], c=cluster.labels_)
```

Desde otra perspectiva:

```
plt.figure(figsize=(10, 7))
plt.scatter(data_scaled['Fresh'], data_scaled['Frozen'], c=cluster.labels_)
```

Otro ejemplo con Python

Importar bibliotecas:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Leer en conjunto de datos:

```
# http://dicyg.fi-c.unam.mx:8080/lalo/pypcd/presentaciones/Mall_Customers.csv
dataset = pd.read_csv('https://bit.ly/2lNwP85')
dataset.head()
X = dataset.iloc[:, [3, 4]].values
y = dataset.iloc[:, 3].values
```

Este conjunto contiene el identificador del cliente, su género y edad; además del ingreso anual y un nivel de consumo. El objetivo es agrupar a los clientes de acuerdo al ingreso anual y el nivel de consumo.

Se realiza el dendrograma para determinar el número óptimo de grupos:

```
import scipy.cluster.hierarchy as sch
dendrograma = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrograma')
plt.xlabel('Clientes')
plt.ylabel('Distancia euclideana')
plt.show()
```

Se obtiene el agrupamiento usando 5 para el número de grupos:

```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean',
                             linkage = 'ward')
y_hc = hc.fit_predict(X)
```

Finalmente se muestran los grupos obtenidos:

```
plt.scatter(X[y_hc == 0,0],X[y_hc == 0,1],s=100,c='red',label='Cluster 1')
plt.scatter(X[y_hc == 1,0],X[y_hc == 1,1],s=100,c='blue',label='Cluster 2')
plt.scatter(X[y_hc == 2,0],X[y_hc == 2,1],s=100,c='green',label='Cluster 3')
plt.scatter(X[y_hc == 3,0],X[y_hc == 3,1],s=100,c='cyan',label='Cluster 4')
plt.scatter(X[y_hc == 4,0],X[y_hc == 4,1],s=100,c='magenta',label='Cluster 5')
plt.title('Grupos de clientes')
plt.xlabel('Ingreso anual (k$)')
plt.ylabel('Nivel de consumo (1-100)')
plt.legend()
plt.show()
```

Programa:

- ♢ Implementar los algoritmos de k-medias y k-modas desde cero
- ♢ Para el agrupamiento aglomerativo:
  - Modificar los ejemplos realizados en clase utilizando la distancias *Manhattan* y *cosine* (ver la documentación) y comparar los resultados
  - Modificar los ejemplos realizados en clase para usar los otros tipos de enlazamiento entre grupos (*single*, *complete*, *average*) y comparar los resultados

*

1.2.3.4. Modelos de mezclas gausianas (*Gaussian Mixed Models*)

Similar a k-medias, en estos modelos se establece una cantidad k de grupos para iniciar y se inicializan tanto la medias como las varianzas.

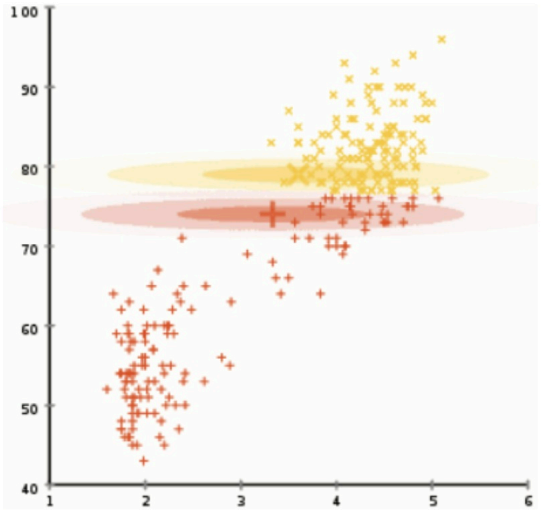


Figura 1.6: Paso inicial de un modelo de mezclas gaussianas

Iterativamente actualiza la media y la varianza de cada grupo, así como la probabilidad de pertenencia de cada punto a cada grupo y el peso del mismo (cantidad de muestras de cada grupo).

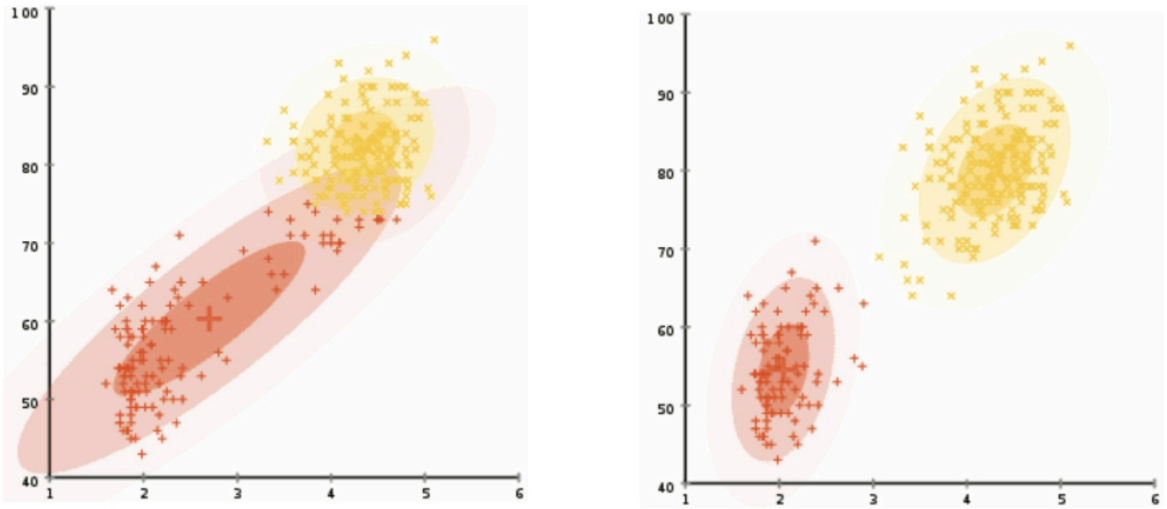


Figura 1.7: Evolución de un modelo de mezclas gaussianas

Esperanza-Maximización (*Expectation-Maximization*), es la técnica más popular para determinar los parámetros de un modelo de mezclas gaussianas. Se compone de dos pasos:

♢ Paso E: asignar de forma probabilística cada muestra a cada clase, basándose en la hipótesis actual de los parámetros.

♢ Paso M: actualizar las hipótesis de los parámetros, en función de las asignaciones actuales.

En el paso E se calcula el valor esperado de las asignaciones de grupos. En el paso M se obtiene la nueva máxima verosimilitud de las hipótesis.

Modelos de mezclas gaussianas vs k-medias

Si bien el algoritmo inicia con un número predefinido de grupos al igual que k-means, existen diferencias fundamentales:

♢ k-medias establece un círculo (hiperesfera) al centro del grupo, con radio definido por el punto más distante

♢ GMM funciona mejor cuando los datos no se ajustan a circunferencias (hiperesferas)

♢ k-medias realiza clasificación dura: devuelve la clase a la que pertenece una muestra, mientras GMM hace clasificación suave: devuelve la probabilidad de pertenencia a cada clase.

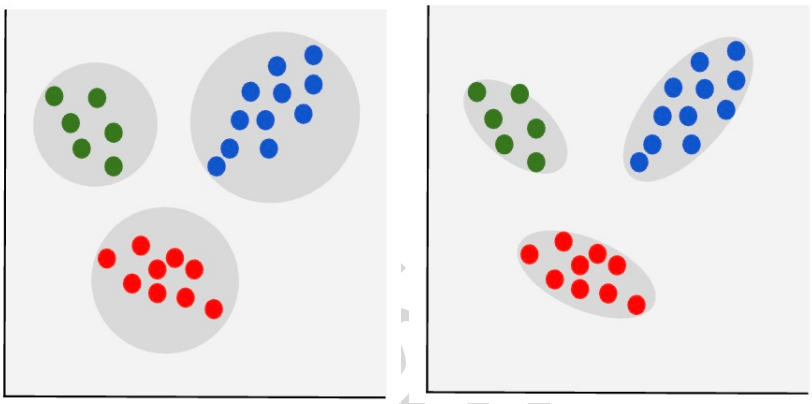


Figura 1.8: k-medias vs GMM

Ejemplo:

Importando bibliotecas:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.datasets.samples_generator import make_blobs
```

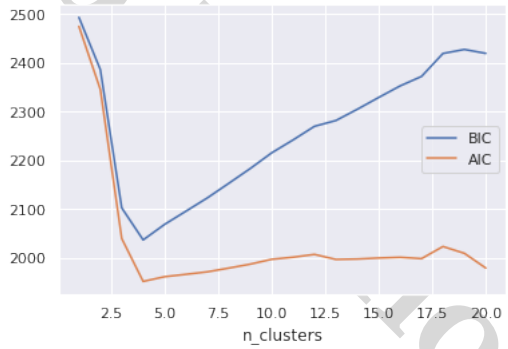
Generamos conjuntos de prueba:

```
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.6, random_state=0)
plt.scatter(X[:,0],X[:,1])
```

Antes de aplicar el modelo, podemos obtener el número óptimo de grupos con ayuda de los criterios:

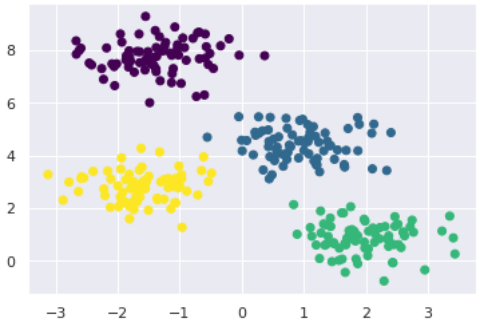
- ♢ *Akaike information criterion*(AIC)
- ♢ *Bayesian information criterion*(BIC)

```
n_clusters = np.arange(1, 21)
models = [GaussianMixture(n, covariance_type='full',
                          random_state=0).fit(X) for n in n_clusters]
plt.plot(n_clusters, [m.bic(X) for m in models], label='BIC')
plt.plot(n_clusters, [m.aic(X) for m in models], label='AIC')
plt.legend(loc='best')
plt.xlabel('n_clusters')
```



En este caso el criterio indica elegir el valor que minimiza el criterio elegido y para nuestro ejemplo coincide para ambos.

```
# Objeto GaussianMixture con 4 grupos
gmm = GaussianMixture(n_components=4)
# Ajuste
gmm.fit(X)
# Predicción
labels = gmm.predict(X)
# Centroides
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis');
```



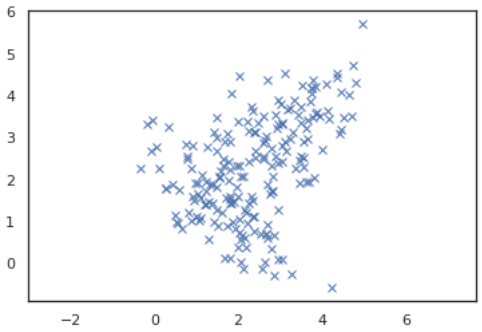
Ejemplo 2:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
```

Datos:

```
#https://bit.ly/3Bic3iu
X_train = np.load('data.npy')
```

```
plt.plot(X_train[:,0], X_train[:,1], 'bx')
plt.axis('equal')
plt.show()
```

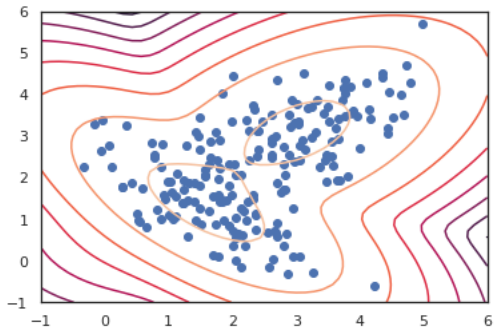


Modelo y resultados:

```
gmm = GaussianMixture(n_components=2) gmm.fit(X_train)

print("Medias: \n", gmm.means_)
print("Covarianzas: \n",gmm.covariances_)
```

Gráfica de los grupos con curvas de nivel:

```
X, Y = np.meshgrid(np.linspace(-1, 6),
np.linspace(-1,6))
XX = np.array([X.ravel(), Y.ravel()]).T
Z = gmm.score_samples(XX)
Z = Z.reshape((50,50))
plt.contour(X, Y, Z)
plt.scatter(X_train[:, 0], X_train[:, 1])
plt.show()
```



1.2.3.5. DBSCAN

Density Based Spacial Clustering of Applications with Noise, es un algoritmo de agrupamiento basado en densidad: define los grupos como regiones continuas de alta densidad.

Esta técnica clasifica los puntos como **punto núcleo** (*core points*) (densamente alcanzables) o **anomalías** (*noise points*) de la forma siguiente:

♢ Para cada elemento del conjunto de datos, contar cuántos puntos se encuentran a una *distancia pequeña* ϵ, conocida como $\epsilon - vecindario$; si un elemento contiene más de cierto umbral *min* dentro de su $\epsilon - vecindario$, se considera como un **punto núcleo** (*core point*).

♢ Todos los elementos en el $\epsilon - vecindario$ de un punto núcleo pertenecen al mismo grupo, puede incluir otros *core points*. Los puntos alcanzables que no tienen más de *min* elementos en su $\epsilon - vecindario$ son llamados **puntos frontera** (*border points*).

♢ Cualquier elemento que no pertenezca a algún grupo, se considera como una **anomalía** o ruido (*noise point*).

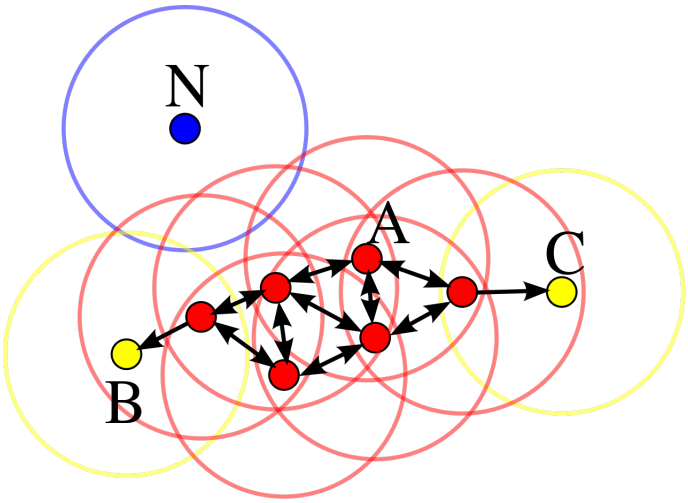


Figura 1.9: DBSCAN: Los puntos rojos son puntos núcleo; los puntos amarillos son puntos frontera alcanzables y pertenecen al mismo grupo; el punto azul no es alcanzable y se considera como una anomalía. (https://bit.ly/3Dy7qlf)

Una de las ventajas de DBSCAN es que no requiere que se especifique el número de grupos a generar; pero su principal atractivo es que puede agrupar los puntos con formas geométricas arbitrarias.

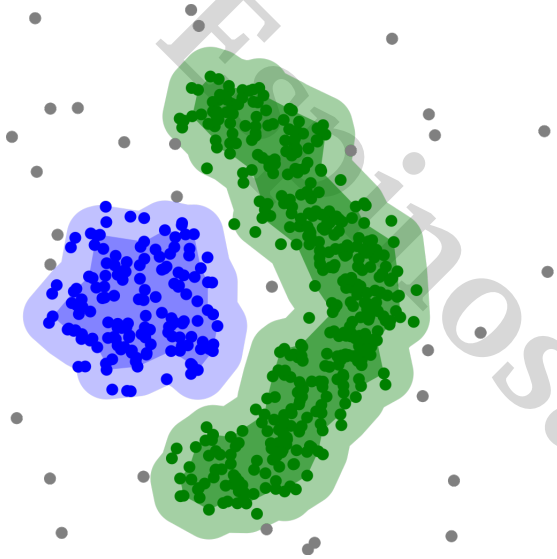


Figura 1.10: DBSCAN puede determinar grupos con formas complicadas. (https://bit.ly/3v3kb46)

Elección de los parámetros

♢ El valor para *min* puede ser obtenido a partir de la dimensionalidad de los puntos p como $min \geq p+1$; cuanto más grande sea el conjunto de datos, mayor debe ser el valor asignado a *min*.

♢ Idealmente, el valor ϵ está dado por el problema a resolver, p.e. una distancia física. Cuando esto no sea el caso, debe tenerse en cuenta que un valor pequeño de ϵ puede provocar que muchos datos no se agrupen, mientras que con un valor grande los grupos se fusionarán y la mayoría de los elementos podrían quedar dentro de un solo grupo.

Ejemplo:

Importando bibliotecas:

```
import pandas as pd
from pylab import rcParams
import seaborn as sb
import matplotlib.pyplot as plt
import numpy as np
import sklearn
from sklearn.cluster import DBSCAN
from collections import Counter
rcParams['figure.figsize'] = 5, 4
```

Datos, 6463 muestras:

```
df = pd.read_csv("https://bit.ly/3arouNg")
df.head(3)
```

Seleccionamos *fixed acidity* y *volatile acidity*:

```
data = df.iloc[:,1:3]
data.head(3)
```

Creamos y entrenamos un modelo DBSCAN:

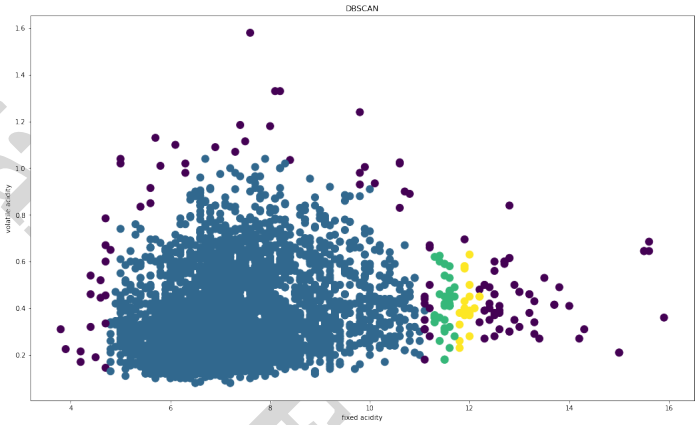
```
model = DBSCAN(eps=0.2, min_samples=20).fit(data)
print(model)
```

Mostramos los resultados, la etiqueta -1 representa a los puntos anómalos:

```
print(Counter(model.labels_))
outliers_df = pd.DataFrame(data)
print(outliers_df[model.labels_==-1])
```

Visualizamos los resultados:

```
fig = plt.figure()
ax = fig.add_axes([.1,.1, 2.3,2])
colors = model.labels_
ax.scatter(data.iloc[:,0].values,data.iloc[:,1].values,c=colors, s=120)
ax.set_xlabel('fixed acidity')
ax.set_ylabel('volatile acidity')
plt.title('DBSCAN')
```



Tarea:

♢ Aplica los modelos de agrupamiento presentados a los conjuntos Iris y Digits; compara los resultados de cada uno de ellos.

*