islfd

Plan

- R Programming
- Probabilidad y Estadística (crash course)
- Aprendizaje Estadístico
- Ejemplo práctico

R Programming

- Propósito y Task Views
- Instalación de paquetes
- Ayuda
- Tipos de datos
- Tipos de objeto
- Operaciones básicas
- Estructuras de control
- Environments
- Vectorización, funciones anónimas
- Paquetes útiles
- Graficación: ggplot2

Propósito y Task Views

R es un lenguaje creado para hacer cómputo estadístico y graficación. Ha sido ampliamente adoptado y extendido por miembros de la comunidad científica, y tiene performance comparable a MATLAB o GNU Octave en cuanto a cómputo con matrices.

La más grande ventaja de R es el acceso a cientos de paquetes a través del CRAN, con mirrors alrededor del mundo. El CRAN define y mantiene Task Views, listas de paquetes con un propósito en común. También mantiene versiones binarias de muchos paquetes para Windows y OS X.

Algunas Task Views

- Bayesian Inference
- Clinical Trial Design, Monitoring and Analysis
- Econometrics
- Analysis of Ecological and Environmental Data
- Empirical Finance
- Statistical Genetics
- Natural Language Processing
- Machine Learning & Statistical Learning
- Analysis of Spatial Data
- Graphical Models

Full list

Instalación de paquetes

Para instalar paquetes del CRAN

```
install.packages("dplyr")
install.packages("file:///tmp/dplyr.tgz", repos=NULL)
```

Si no hay una versión binaria disponible, `install.packages` compilará las fuentes indicadas, también se encarga de resolver dependencias de paquetes.

Para instalar todos los paquetes en un task view

```
install.packages("ctv")
library("ctv")
install.views("NaturalLanguageProcessing")
```

Instalación de paquetes

El paquete `devtools` nos permite instalar un paquete directamente desde su fuente en github, bitbucket, svn, etc.

```
install.packages("devtools")
library("devtools")
install_github("hadley/ggplot2")
```

El IDE RStudio ofrece una manera sencilla de realizar estas instalaciones, además de ofrecer integración con `packrat`, un paquete que permite aislar el ambiente de R usando una biblioteca privada de paquetes (similar a `virtualenv`).

Ayuda

R ofrece una manera de acceder a la documentación de una función a través de ?. Es común que las funciones que un paquete expone estén documentadas, así como encontrar documentación de un paquete en general

```
?library
?devtools
```

También podemos obtener ayuda acerca de un operador

```
?`+`
```

Ayuda

Además, algunos paquetes con funcionalidad compleja ofrecen vignettes, scon quickstarts o tutoriales acerca de su uso

```
vignette(package = "dplyr")
vignette("databases", package = "dplyr")
```

Si todo lo demás falla, existe una comunidad activa en stack overflow para todo lo relacionado con R como lenguaje, así como cross validated para todas las preguntas de índole estadística.

Tipos de datos

Existen 6 tipos de datos "atómicos" en R

```
class(TRUE)
```

```
## [1] "logical"
```

```
class(42)
```

```
## [1] "numeric"
```

```
class(42L)
```

```
## [1] "integer"
```

Tipos de datos

Existen 6 tipos de datos "atómicos" en R

```
class(4 + 2i)
```

```
## [1] "complex"
```

```
class("42")
```

```
## [1] "character"
```

```
class(charToRaw("B"))
```

```
## [1] "raw"
```

Tipos de datos

Adicionalmente, un factor puede ser usado para expresar variables categóricas. Un factor puede ser ordenado.

```
class(factor(1))
```

```
## [1] "factor"
```

Tipos de datos

NULL representa al objeto nulo. Indica que un valor o función no está definida

```
is.null(NULL)
```

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

```
is.null(NA)
```

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

```
1 == NULL
```

```
## logical(0)
```

Tipos de datos

`NA` representa datos faltantes, es una constante de longitud 1. `NA` puede ser cualquier tipo de dato excepto `raw`. `NA` no es `NULL`, y existen comportamientos predefinidos para lidiar con datos faltantes.

```
NA + 1
```

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

```
sum(c(1, 2, 3, NA), na.rm = TRUE)
```

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

```
na.fail(c(1, 2, 3, NA))
```

```
## Error in na.fail.default(c(1, 2, 3, NA)): missing values in o
```

Tipos de objeto

Existen muchos tipos de objeto en R, y dos maneras comunes de definir tipos de objeto (clases S3 y S4). Sin embargo, los tipos más comunes son vectores, listas,matrices y data frames.

No existe el tipo de objeto "escalar" en R, el dato atómico es el vector.

```
length(1)
```

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

Vectores

Un vector sólo puede contener un tipo de dato. Si un elemento de otro tipo se concatena a un vector, el vector resultante es del tipo más general. La manera más fácil de construir vectores es a través de c

```
c(TRUE, FALSE)
```

```
## [1]  TRUE FALSE
```

```
c(TRUE, FALSE, 2)
```

```
## [1] 1 0 2
```

```
c(TRUE, FALSE, 2, "")
```

```
## [1] "TRUE"  "FALSE" "2"     ""
```

Vectores

Se puede acceder de manera posicional a los elementos de un vector

```
v <- 10:13
v[[1]]
```

```
## [1] 10
```

```
v[c(1:2)]
```

```
## [1] 10 11
```

Vectores

Los elementos de un vector pueden ser nombrados y también accedidos de ese modo

```
names(v) <- c("uno", "dos", "tres")
v[["uno"]]
```

```
## [1] 10
```

Listas

Una lista puede contener cualquier tipo de objeto y combinar distintos tipos de datos.

```
l <- list(1, TRUE, "...")
l
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] "..."
```

Listas

Al igual que un vector, se puede acceder de manera posicional o a través de nombre. A diferencia de un vector, se puede acceder usando el operador $

```
names(l) <- c("uno", "true", "ellipsis")
l$uno
```

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

Listas

Se pueden borrar elementos de una lista refiriendolos a NULL

```
l$true <- NULL
l
```

```
## $uno
## [1] 1
##
## $ellipsis
## [1] "..."
```

Matrices

Una matriz es como un vector, aunque con dos dimensiones. Las matrices se llenan y se despliegan en orden de columnas.

```
m <- matrix(c(1,2,3,4), nrow = 2)
m
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
m[1, 2]
```

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

Al igual que un vector, sólo pueden contener un tipo de dato

Matrices

Las columnas y las filas se pueden nombrar por separado

```
colnames(m) <- c("col1", "col2")
rownames(m) <- c("row1", "row2")
m
```

```
##      col1 col2
## row1    1    3
## row2    2    4
```

```
m["row1", "col1"]
```

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

Matrices

`dim`, `nrow` y `ncol` nos dan las dimensiones de una matriz

```
dim(m)
```

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

```
nrow(m)
```

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

```
ncol(m)
```

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

Data Frames

Un data frame es una colección de variables, que se comporta como matriz o como lista de columnas, según la función que se le aplique.

```
df <- data.frame(col1 = c(1,2), col2 = c("tres","cuatro"))
df
```

```
##   col1   col2
## 1    1   tres
## 2    2 cuatro
```

Un data frame puede contener distintos tipos de datos

Data Frames

```
dim(df)
```

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

```
df$col1
```

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

```
names(df)
```

```
## [1] "col1" "col2"
```

Factores

Un tipo de objeto adicional es el factor, que sirve para representar variables categóricas, ordenadas o no

```
f <- factor(c("A veces", "Siempre", "A veces", "Nunca"),
            levels = c("Nunca", "A veces", "Siempre"))
f
```

```
## [1] A veces Siempre A veces Nunca
## Levels: Nunca A veces Siempre
```

```
as.ordered(f)
```

```
## [1] A veces Siempre A veces Nunca
## Levels: Nunca < A veces < Siempre
```

Factores

```
nms <- c("John", "Paul", "George", "Ringo")
df <- data.frame(nombre=nms, sexo=f, stringsAsFactors = F)
df
```

```
##   nombre    sexo
## 1   John A veces
## 2   Paul Siempre
## 3 George A veces
## 4  Ringo   Nunca
```

```
summary(df)
```

```
##     nombre              sexo
##  Length:4           Nunca  :1
##  Class :character   A veces:2
##  Mode  :character   Siempre:1
```

Operaciones básicas

```
7 + 2
```

```
## [1] 9
```

```
7 - 2
```

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

```
7 * 2
```

```
## [1] 14
```

Operaciones básicas

```
7 / 2
```

```
## [1] 3.5
```

```
7 %% 2
```

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

```
7 %/% 2
```

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

Operaciones básicas

```
7 / 2
```

```
## [1] 3.5
```

```
7 ^ 2
```

```
## [1] 49
```

```
7 ** 2
```

```
## [1] 49
```

Operaciones básicas

```
7 %/% 2
```

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

```
7 %% 2
```

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

Operaciones básicas

```
TRUE & FALSE
```

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

```
TRUE | FALSE
```

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

```
!TRUE
```

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

Operaciones básicas

Todas estas operaciones se pueden aplicar a vectores, y se aplicarán componente a componente, repitiendo el vector más pequeño como sea necesario

```
c(1, 2) + c(3, 0)
```

```
## [1] 4 2
```

```
c(2, 1) * c(2, 2)
```

```
## [1] 4 2
```

```
c(1, 2, 3, 4) ^ c(2, 3)
```

```
## [1]  1  8  9 64
```

Estructuras de control

`if`, `else`, `for`, `while`, `break`, `next` funcionan como en otros lenguajes.

```
for(i in 1:5) {
  if(i < 3) {
    print(i)
  } else {
    print(6 - i)
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 2
## [1] 1
```

Funciones

Para definir una función

```
do.something <- function(x, y) {
  x + y
}
do.something(10, 3)
```

```
## [1] 13
```

Funciones

Valores por defecto en argumentos

```
do.something <- function(x, y=1) {
  x + y
}
do.something(10)
```

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

```
do.something(y=3, 10)
```

```
## [1] 13
```

Funciones

Argumentos adicionales, no definidos

```
do.something <- function(x, y=1, ...) {
  x + y + sum(...)
}
do.something(10, z=3, a=5)
```

```
## [1] 19
```

`...` sólo puede ser usado como argumento de una función

Funciones

`do.call` llama la función especificada con una lista de argumentos

```
args <- list(x=4, y=3, z=1)
do.call(do.something, args)
```

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

```
do.call("do.something", args)
```

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

Environments

Un environment es un espacio de nombres, usado en las reglas de scoping de R. (Scope léxico) También se pueden crear manualmente.

```
e <- new.env()
e$a <- FALSE
e$b <- "b"
e$uno <- 1
ls(e)
```

```
## [1] "a"    "b"    "uno"
```

Environments

Un environment se comporta de manera similar a una lista, pero difiere de varias formas importantes. No se puede acceder posicionalmente a nombres de un environment. Los environments no están ordenados

```
l[[1]]
```

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

```
e[[1]]
```

```
## Error in e[[1]]: wrong arguments for subsetting an envi
```

Environments

Una lista puede contener varios elementos con el mismo nombre, aunque esto no es recomendado

```
list(a="a", a="b")
```

```
## $a
## [1] "a"
##
## $a
## [1] "b"
```

```
e$a <- "b"
e$a
```

```
## [1] "b"
```

Environments

Un environment tiene un "padre". Solo el environment vacío no tiene un padre.

```
parent.env(e)
```

```
## <environment: R_GlobalEnv>
```

```
parent.env(globalenv())
```

```
## <environment: package:stats>
## attr(,"name")
## [1] "package:stats"
## attr(,"path")
## [1] "/usr/lib/R/library/stats"
```

Environments

Asignar un elemento de una lista a `NULL` lo elimina de la lista. Hacerlo en un environment crea una referencia a `NULL`.

```
e$a <- NULL
ls(e)
```

```
## [1] "a"   "b"   "uno"
```

Environments

`search()` regresa la lista de padres del environment global, este es el orden en el que se resuelven los nombres

```
search()
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:graphics
## [4] "package:grDevices" "package:utils"     "package:datasets
## [7] "package:methods"   "Autoloads"         "package:base"
```

```
pryr::where("var")
```

```
## <environment: package:stats>
## attr(,"name")
## [1] "package:stats"
## attr(,"path")
## [1] "/usr/lib/R/library/stats"
```

Environments

Cuando se carga un paquete nuevo, se agrega a `search()` entre el `globalenv()` y el siguiente elemento

```
search()
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:graphics
## [4] "package:grDevices" "package:utils"     "package:datasets
## [7] "package:methods"   "Autoloads"         "package:base"
```

```
library(pryr)
search()
```

```
##  [1] ".GlobalEnv"        "package:pryr"      "package:stats"
##  [4] "package:graphics"  "package:grDevices" "package:utils"
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

Environments

Para checar si un nombre existe en un environment se usa `exists`. Por defecto se busca en todos los padres, para evitarlo usamos `inherits = FALSE`

```
exists("mean", envir=e)
```

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

```
exists("mean", inherits=F)
```

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

Environments

Una función tiene cuatro environments asociados. El enclosing, que es el environment donde fue creada la función, se usa para el scoping. Todos los objetos a los que haga referencia la función se buscaran primero en este environment.

```
environment(sd)
```

```
## <environment: namespace:stats>
```

Environments

Un binding environment es donde existe un nombre que haga referencia a la función, una función puede tener más de un binding environment

```
sd <- stats::sd
pryr::where("sd")
```

```
## <environment: R_GlobalEnv>
```

Environments

Estos dos environments pueden ser distintos, y sólo el primero se usa para las reglas de scoping, asegurando que la función siempre busque las referencias adecuadas

```
var <- "Can't break this"
pryr::where("var")
```

```
## <environment: R_GlobalEnv>
```

```
sd(c(1, 2, 3))
```

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

```
environment(sd)
```

```
## <environment: namespace:stats>
```

```
pryr::where("sd")
```

```
## <environment: R_GlobalEnv>
```

Environments

El environment de ejecución es efímero, se crea al ejecutar una función, y se destruye al terminar la misma, a menos de que la función regrese a su vez una función

```
plus <- function(x) {
  function(y) x + y
}
plus.1 <- plus(1)
environment(plus.1)
```

```
## <environment: 0x462cb38>
```

```
identical(parent.env(environment(plus.1)), environment(plus))
```

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

Environment

El último environment asociado a una función es el de llamada. Este se puede acceder a través de `parent.frame`, típicamente no será necesario acceder a él, a menos de que se quiera usar evaluación no estándar o simular un scoping dinámico

```
f <- function() {
  x <- 10
  function() {
    print(get("x", environment()))
    print(get("x", parent.frame()))
  }
}
x <- 5
f()()
```

```
## [1] 10
## [1] 5
```

Attach, Detach

`attach` añade un objeto a `search()`. Este objeto puede ser un environment, lista o data frame, permitiendo acceso a su lista de nombres en el environment en el que se esta trabajando.

```
attach(e)
search()
```

```
##  [1] ".GlobalEnv"        "e"                 "package:pryr"
##  [4] "package:stats"     "package:graphics"  "package:grDevic
##  [7] "package:utils"     "package:datasets"  "package:methods
## [10] "Autoloads"         "package:base"
```

```
detach(e)
```

Attach, Detach

`library` funciona como `attach`, para quitar un paquete podemos usar `detach`

```
library(pryr)
detach("package:pryr", unload=T)
search()
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:graphics
## [4] "package:grDevices" "package:utils"     "package:datasets
## [7] "package:methods"   "Autoloads"         "package:base"
```

Vectorización

Muchas funciones base en R incluyen llamadas a `.Internal`, `.Primitive` o `.Call`

```
sum
```

```
## function (..., na.rm = FALSE)  .Primitive("sum")
```

```
paste
```

```
## function (..., sep = " ", collapse = NULL)
## .Internal(paste(list(...), sep, collapse))
## <bytecode: 0x244d3a8>
## <environment: namespace:base>
```

Vectorización

Estas funciones de hecho son llamadas a rutinas de C o Fortran, por razones de velocidad. Esto a menudo quiere decir que es más rápido llamar una función predefinida que replicarla

```
for.sum <- function(m) {
  s <- 0
  for(i in v) s <- s + i
  s
}
```

Vectorización

```
v <- matrix(rbinom(1e6, 100, 0.5), ncol = 1e3)
system.time(rep(for.sum(v), 1e6))
```

```
##    user  system elapsed
##   0.152   0.000   0.153
```

```
system.time(sum(v, 1e6))
```

```
##    user  system elapsed
##   0.000   0.000   0.001
```

Para reducir el overhead de llamar repetidamente las funciones de C, es útil pensar en aritmética de vectores y en utilizar funciones primitivas, en lugar de escribir ciclos `for`. Esto es especialmente un problema si R tiene que cambiar el tamaño de un objeto en cada paso del ciclo

Vectorización

Otra manera de "vectorizar" es utilizar funciones de la familia `apply`. En este caso no siempre se están reduciendo las llamadas a funciones internas, pero se está asegurando que no haya efectos secundarios y el código se vuelve más leíble

```
apply(v, MARGIN=2, FUN=mean)[1:5]
```

```
## [1] 50.034 49.882 49.947 49.986 50.168
```

Vectorización

```
m <- nrow(v)
n <- ncol(v)
means <- rep(0, n)
for(j in 1:n) {
  col.sum <- 0
  for(i in 1:m) {
    col.sum <- col.sum + v[i, j]
  }
  means[j] <- col.sum/m
}
means[1:5]
```

```
## [1] 50.034 49.882 49.947 49.986 50.168
```

Vectorización

Por último, si la función a aplicar a una secuencia se va a utilizar sólo una vez, y es fácil de leer, se puede utilizar una función anónima.

```
apply(v, MARGIN=2, FUN=function(col) quantile(col, 0.3) )[1:5]
```

```
## [1] 47 48 47 48 48
```

Paquetes útiles

Hay paquetes que mejoran algunos aspectos de R en general, y ayudan a evitar algunas deficiencias de la sintáxis. Todos los paquetes que incluyo en esta sección están en el CRAN.

Magrittr

Magrittr añade un operador con funcionalidad de pipe

```
library(magrittr)
3 %>% sum(1) %>% prod(2)
```

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

```
prod(2, sum(1, 3))
```

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

Magrittr

```
data.frame(a=rnorm(5, 5, 1), b=runif(5)) %>%
  scale() %>%
  summary()
```

```
##        a                    b
##  Min.   :-1.71763   Min.   :-1.6057
##  1st Qu.:-0.01836   1st Qu.:-0.3149
##  Median : 0.44987   Median : 0.4007
##  Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.54314   3rd Qu.: 0.7015
##  Max.   : 0.74298   Max.   : 0.8184
```

Magrittr

Añade también otros operadores con funcionalidad similar pero resultado distinto, por ejemplo, %<>% funciona como %>% pero actualiza el nombre en el lado izquierdo con el resultado de la expresión, ver `?magrittr` para más detalles.

Lubridate, Zoo

`lubridate` y `zoo` añaden mejor soporte para fechas. Aunque R incluye soporte para fechas, estos paquetes lo mejoran.

```
library(zoo)
library(lubridate)
interval(as.Date("2016-01-01"), Sys.Date()) / months(1)
```

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

```
as.Date("2016-01-31") %m+% months(0:4)
```

```
## [1] "2016-01-31" "2016-02-29" "2016-03-31" "2016-04-30" "2016
```

```
as.yearmon(Sys.Date())
```

```
## [1] "jun 2016"
```

Lubridate, Zoo

yearmon además puede ser operado como numérico, con el entero representando un año completo

```
as.yearmon(Sys.Date()) + 1/12
```

```
## [1] "jul 2016"
```

```
as.Date("2016-01-31") %m+% months(0:4) %>% as.yearmon() + 1
```

```
## [1] "ene 2017" "feb 2017" "mar 2017" "abr 2017" "may 2017"
```

Pryr

`pryr` incluye algunas funciones para lidiar con los componentes internos de R (como `where`). De sus funciones más útiles es partial

```
library(pryr)
my.mean <- partial(mean, na.rm=T)
my.mean(c(1, 2, 3, NA, 4))
```

```
## [1] 2.5
```

Jsonlite

`jsonlite` añade la capacidad de leer y serializar JSON a partir de objetos de R.

```
library(jsonlite)
j <- '[{"a": 1, "b": 2, "c": "A"}, {"a": 4, "b": 1, "c": "B"}]'
d <- fromJSON(txt = j)
d
```

```
##   a b c
## 1 1 2 A
## 2 4 1 B
```

```
toJSON(d)
```

```
## [{"a":1,"b":2,"c":"A"},{"a":4,"b":1,"c":"B"}]
```

Dplyr

`dplyr` añade muchas funciones para manipulación de datos a través de data frames, con un gran performance y backends para varias bases de datos. Nos permite manipular directamente tablas de una BD, sin escribir SQL, toda la evaluación es lazy y, de ser posible, se realiza directamente en la BD.

Para estos ejemplos utilizaremos un data frame local. Para saber más de como usar `dplyr` con bases de datos, revisar `vignette("databases", package="dplyr")`

Dplyr

```
library(dplyr)
mtc <- tbl_df(mtcars)
mtc$car <- row.names(mtcars)
head(mtc)
```

```
## Source: local data frame [6 x 12]
##
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am
##    (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
## 1   21.0     6   160   110  3.90 2.620 16.46     0     1
## 2   21.0     6   160   110  3.90 2.875 17.02     0     1
## 3   22.8     4   108    93  3.85 2.320 18.61     1     1
## 4   21.4     6   258   110  3.08 3.215 19.44     1     0
## 5   18.7     8   360   175  3.15 3.440 17.02     0     0
## 6   18.1     6   225   105  2.76 3.460 20.22     1     0
## Variables not shown: car (chr)
```

Dplyr | Select

```
mtc %>% select(car, cyl, mpg) %>%
  head()
```

```
## Source: local data frame [6 x 3]
##
##                  car   cyl   mpg
##                (chr) (dbl) (dbl)
## 1          Mazda RX4     6  21.0
## 2      Mazda RX4 Wag     6  21.0
## 3         Datsun 710     4  22.8
## 4     Hornet 4 Drive     6  21.4
## 5 Hornet Sportabout     8  18.7
## 6            Valiant     6  18.1
```

Dplyr | Filter

```
mtc %>% select(car, cyl, mpg) %>%
  filter(cyl >= 6) %>%
  head()
```

```
## Source: local data frame [6 x 3]
##
##                  car   cyl   mpg
##                (chr) (dbl) (dbl)
## 1          Mazda RX4     6  21.0
## 2      Mazda RX4 Wag     6  21.0
## 3     Hornet 4 Drive     6  21.4
## 4 Hornet Sportabout     8  18.7
## 5            Valiant     6  18.1
## 6         Duster 360     8  14.3
```

Dplyr | Arrange

```
mtc %>% select(car, cyl, mpg) %>%
  filter(cyl >= 6) %>%
  arrange(desc(mpg)) %>%
  head()
```

```
## Source: local data frame [6 x 3]
##
##                  car   cyl   mpg
##                (chr) (dbl) (dbl)
## 1   Hornet 4 Drive     6  21.4
## 2         Mazda RX4     6  21.0
## 3     Mazda RX4 Wag     6  21.0
## 4      Ferrari Dino     6  19.7
## 5          Merc 280     6  19.2
## 6 Pontiac Firebird     8  19.2
```

Dplyr | Mutate

```
mtc %>% select(car, cyl, mpg) %>%
  filter(cyl >= 6) %>%
  arrange(desc(mpg)) %>%
  mutate(kpl = 1.6/3.78*mpg) %>%
  head()
```

```
## Source: local data frame [6 x 4]
##
##                 car   cyl   mpg      kpl
##               (chr) (dbl) (dbl)    (dbl)
## 1   Hornet 4 Drive     6  21.4 9.058201
## 2         Mazda RX4     6  21.0 8.888889
## 3     Mazda RX4 Wag     6  21.0 8.888889
## 4      Ferrari Dino     6  19.7 8.338624
## 5          Merc 280     6  19.2 8.126984
## 6 Pontiac Firebird     8  19.2 8.126984
```

Dplyr | Group by + Summarise

```
mtc %>%
  select(car, cyl, mpg) %>%
  filter(cyl >= 6) %>%
  arrange(desc(mpg)) %>%
  mutate(kpl = 1.6/3.78*mpg) %>%
  group_by(cyl) %>%
  summarise(mean.kpl = mean(kpl), n = n())
```

```
## Source: local data frame [2 x 3]
##
##      cyl mean.kpl     n
##    (dbl)    (dbl) (int)
## 1      6 8.356765     7
## 2      8 6.391534    14
```

Dplyr | Join

```
cyls <- mtc %>% select(car, cyl) %>% sample_n(20)
mpgs <- mtc %>% select(car, mpg) %>% sample_n(20)
inner_join(mpgs, cyls) %>% head()
```

```
## Joining by: "car"
```

```
## Source: local data frame [6 x 3]
##
##             car   mpg   cyl
##           (chr) (dbl) (dbl)
## 1   Merc 240D  24.4     4
## 2    Fiat 128  32.4     4
## 3  Camaro Z28  13.3     8
## 4 AMC Javelin  15.2     8
## 5    Merc 230  22.8     4
## 6  Duster 360  14.3     8
```

Dplyr

Las operaciones que se realizan después de `group_by` se aplican a cada grupo por separado. e.g. `arrange` organiza las filas adentro de cada grupo por las columnas indicadas.

Existen versiones de todos estos verbos que toman cadenas de caracteres en lugar de nombres para indicar las columnas, simplemente hay que agregar un _ (`select_` , `mutate_`, etc).

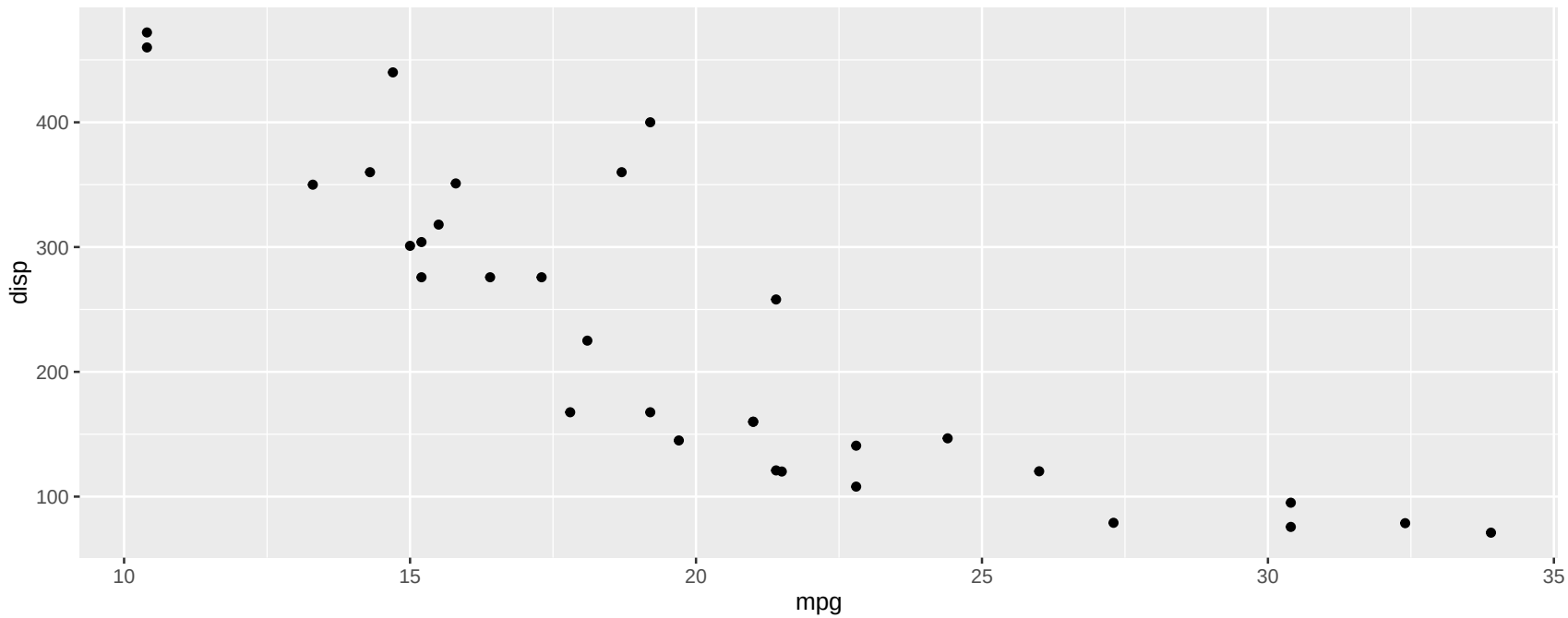
Para una introducción más extensa ver
`vignette("introduction", package="dplyr")`

Ggplot2

`ggplot2` es un sistema de graficación, basado en gramática de gráficas. Los componentes de una gráfica se pueden "sumar" unos a otros, causando overlay de capas o modificaciones, según corresponda

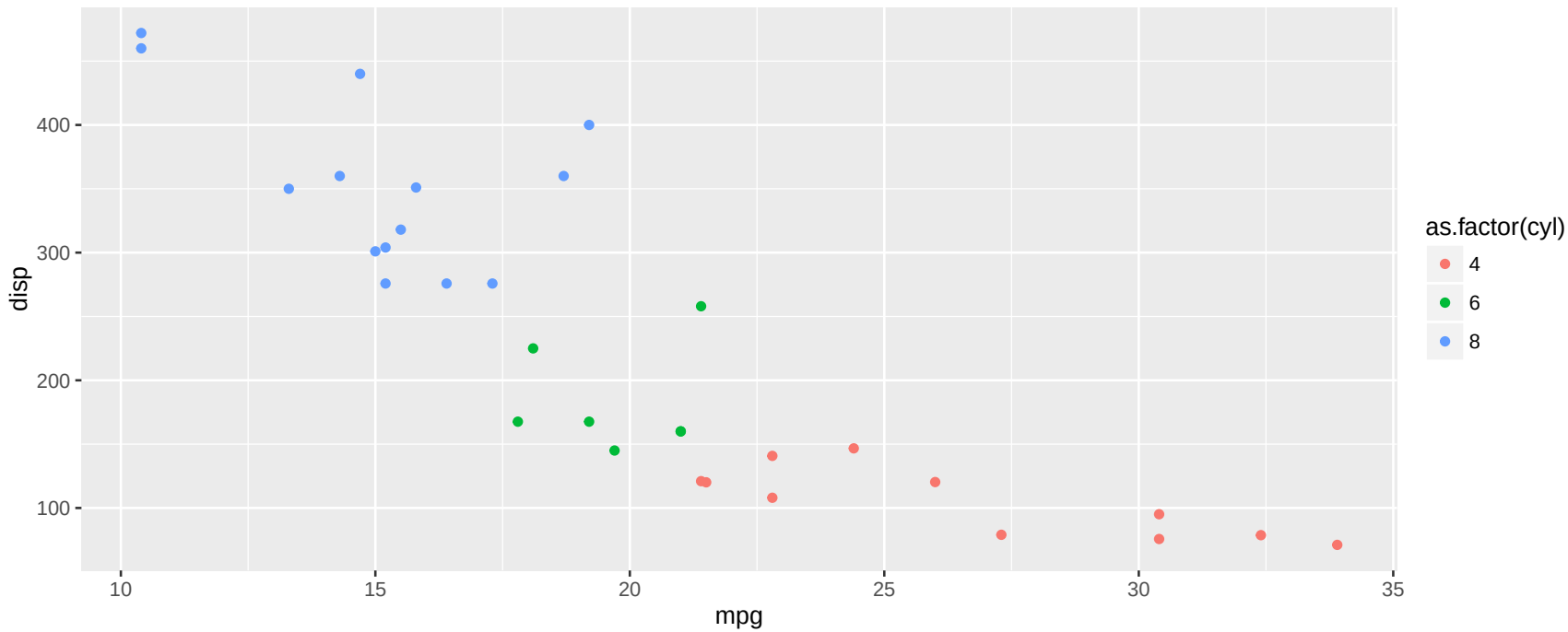
Ggplot2 | Scatter

```
library(ggplot2)
ggplot(mtc, aes(x=mpg, y=disp)) + geom_point()
```



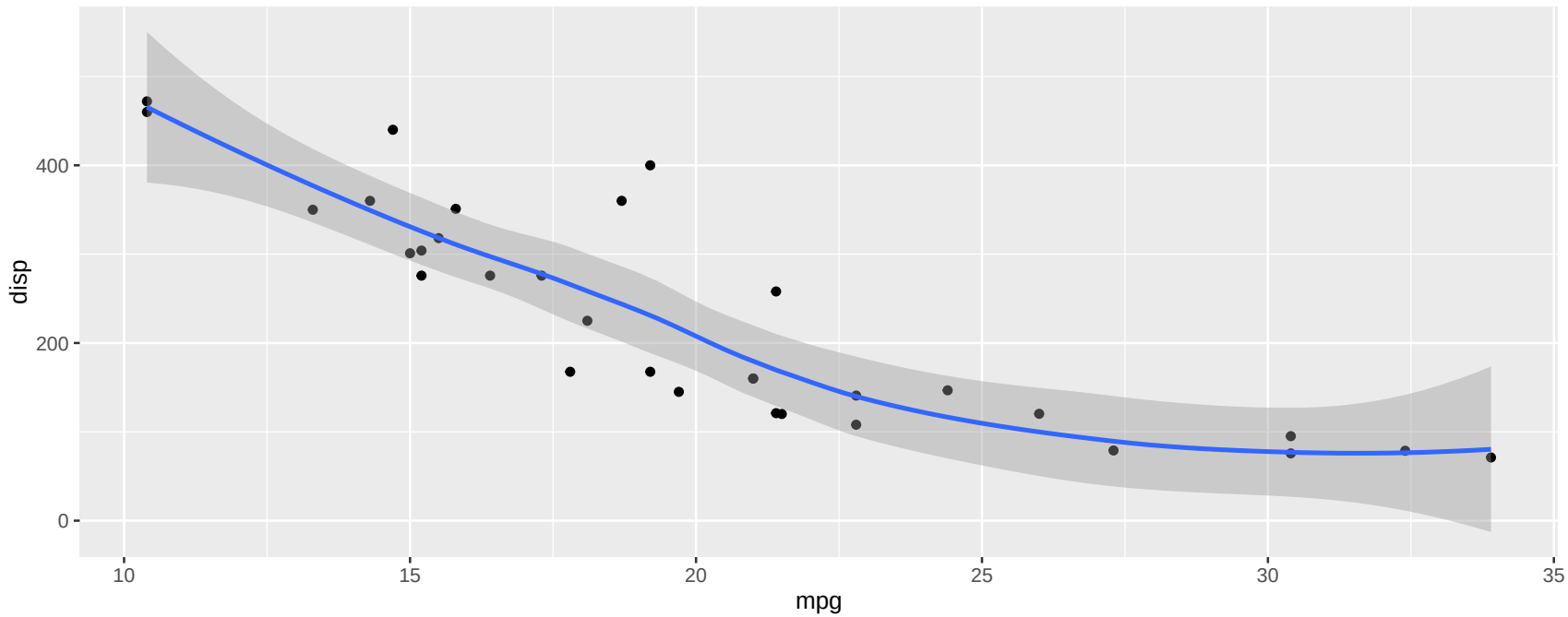
Ggplot2 | Color aesthetic

```
ggplot(mtc, aes(x=mpg, y=disp, color=as.factor(cyl))) + geom_poi
```



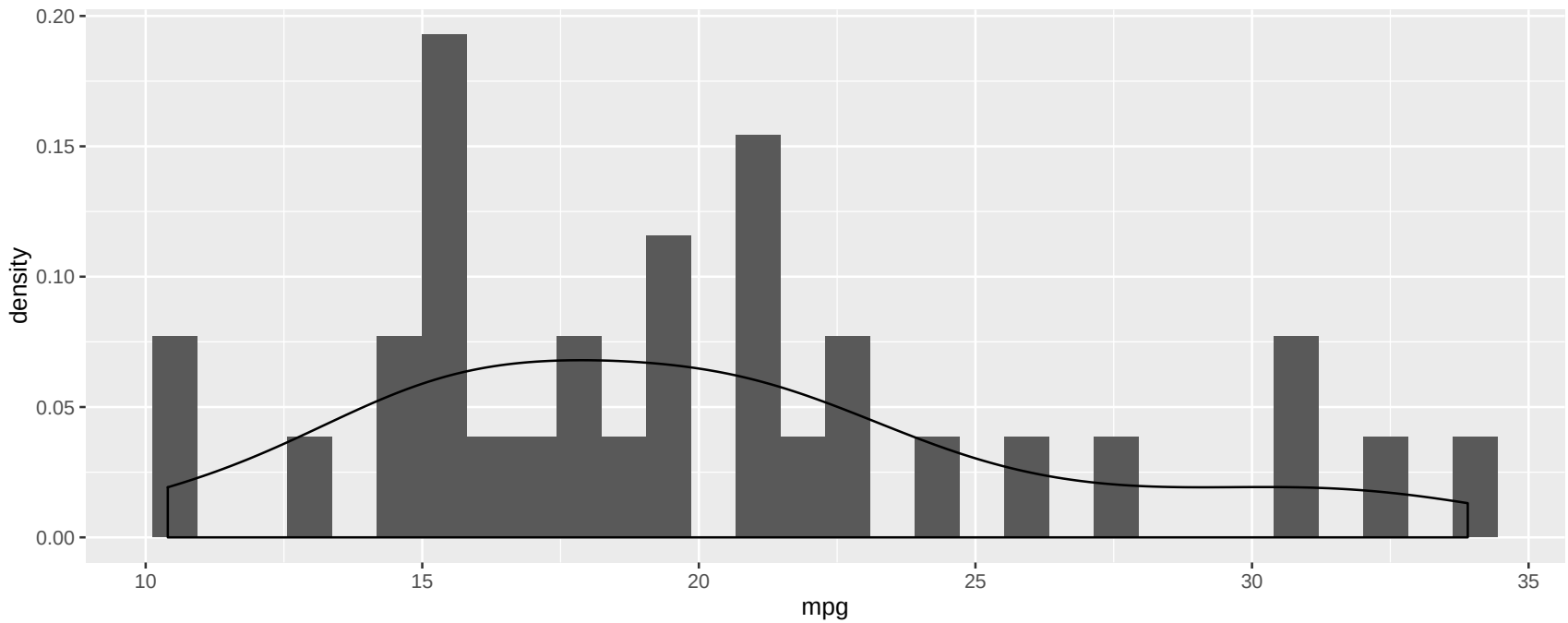
Ggplot2 | Smooth curve (loess)

```
ggplot(mtc, aes(x=mpg, y=disp)) + geom_point() + geom_smooth()
```



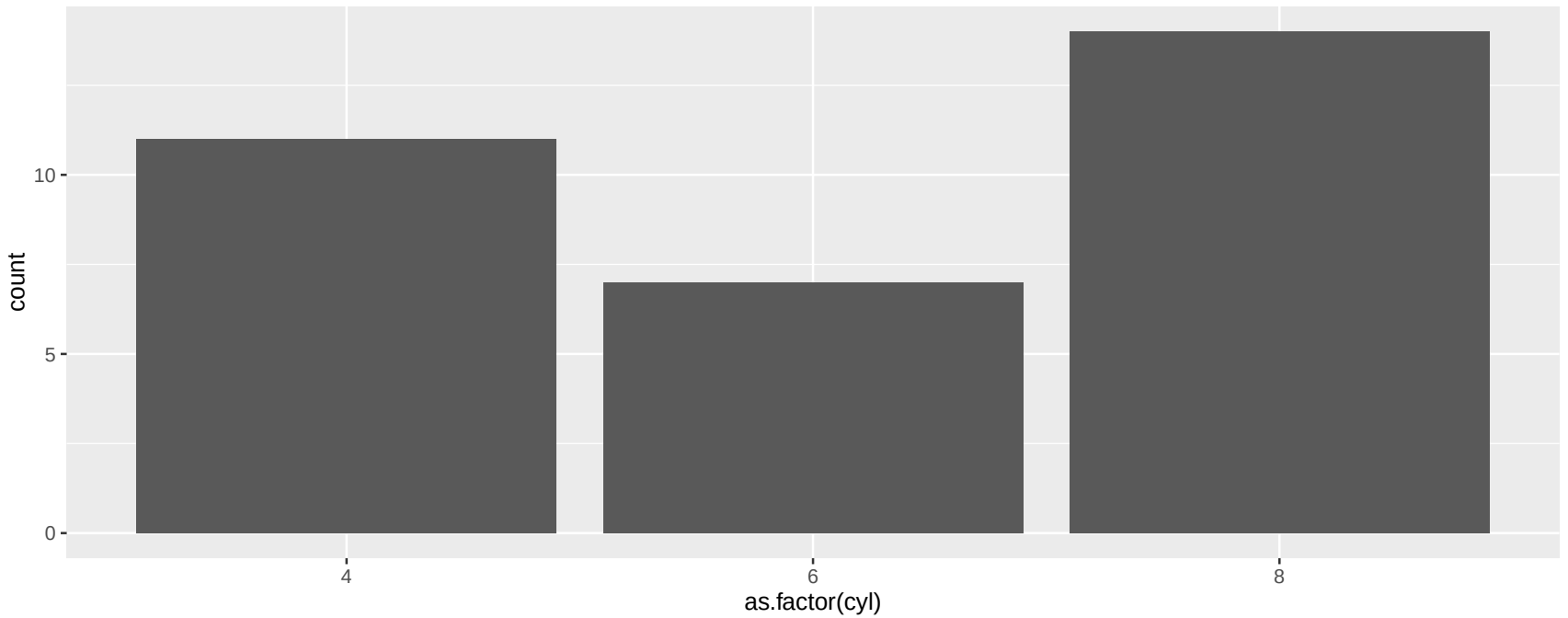
Ggplot2 | Histogram + Density

```
ggplot(mtc, aes(x=mpg, y=..density..)) + geom_histogram() + geom
```



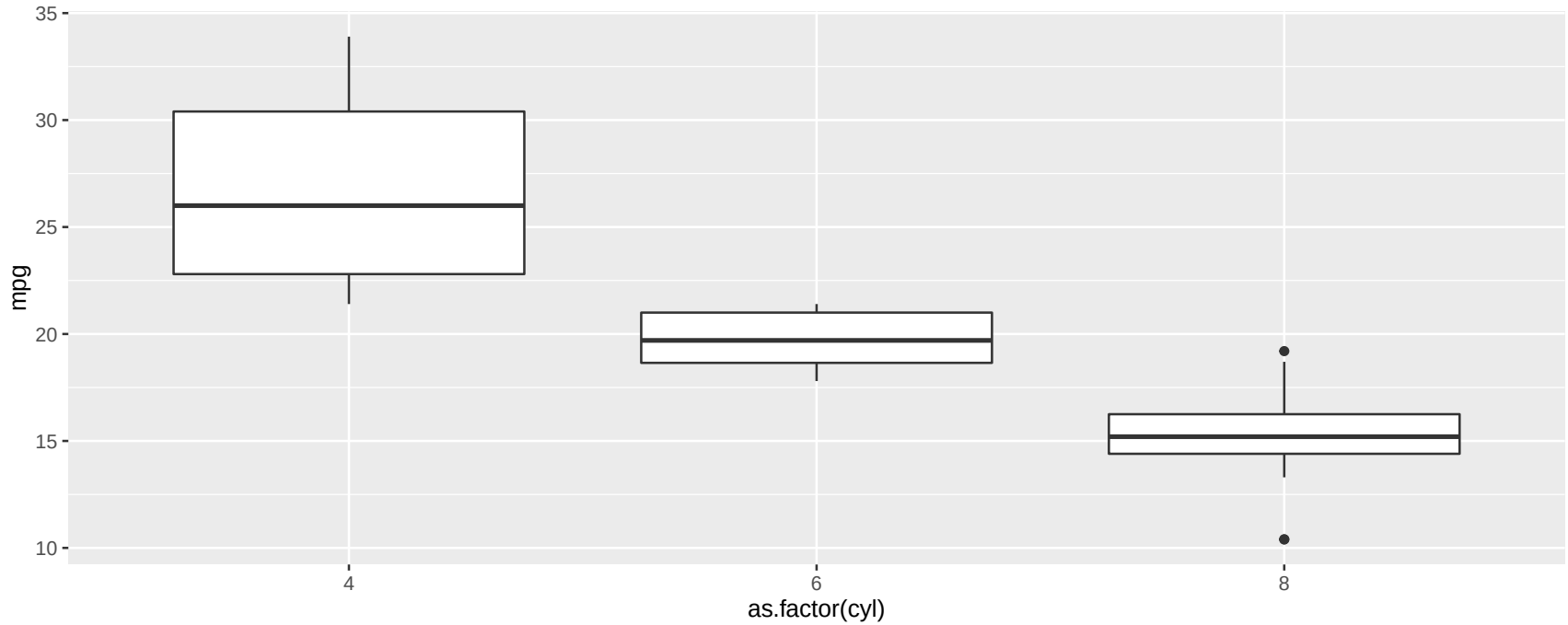
Ggplot2 | Bar (counts)

```
ggplot(mtc, aes(x=as.factor(cyl))) + geom_bar()
```



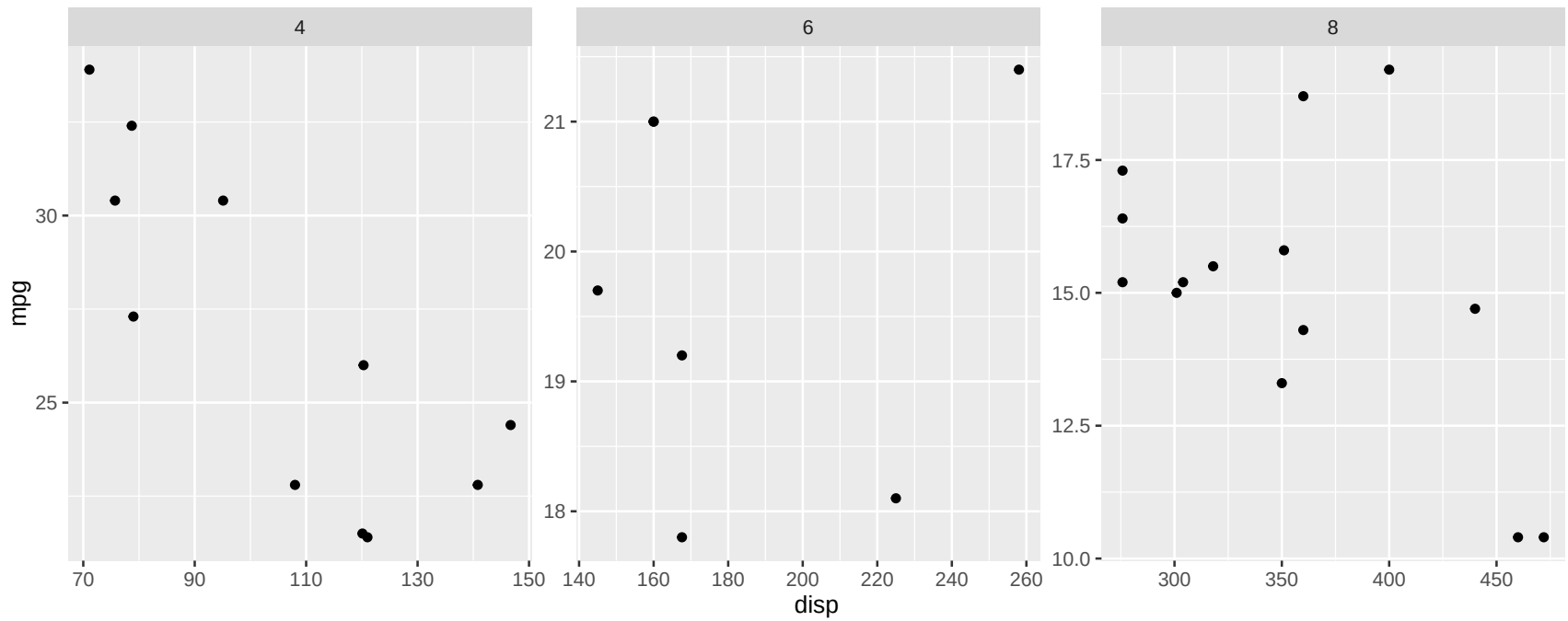
Ggplot2 | Boxplot

```
ggplot(mtc, aes(x=as.factor(cyl), y=mpg)) + geom_boxplot()
```



Ggplot2 | Facet wrap

```
ggplot(mtc, aes(x=disp, y=mpg)) + geom_point() + facet_wrap(~cyl
```



Ggplot2

`ggplot2` también permite controlar leyendas, etiquetas, ejes, etc.
Para más detalles ver la documentación

Probabilidad y Estadística

- Distribución de probabilidad
- Distribución condicional y conjunta
- La distribución normal
- Ley de los grandes números. Implicaciones
- Teorema central de límite. Implicaciones
- Ejemplos de distribuciones
- Estimación de parámetros
- Modelo líneal
- Valores p, R^2
- Interpretaciones de la teoría
- Data Science vs Estadística

Distribución de Probabilidad

Dado un experimento aleatorio con posibles resultados conocidos, una *ley* o *distribución* de probabilidad describe la verosimilitud de cada resultado, asignándole una medida. Los posibles resultados del experimento son codificados en una *variable aleatoria*. El conjunto de posibles valores de dicha variable se conoce como el *soporte* de la distribución.

Si la variable aleatoria es *discreta*, es decir, si toma valores en un conjunto finito o infinito numerable, la distribución se caracteriza por una función *masa de probabilidad*.Si la variable aleatoria es *continua*, se catacteriza por una función de *densidad de probabilidad*.

Distribución de Probabilidad | Distribución Discreta

El resultado de evaluar la función masa en un punto del soporte es igual a la probabilidad de que la variable tome ese valor ($f(x) = Pr(X = x)$), además, la suma de todas esas evaluaciones debe ser igual a 1.

$$\sum_{x \in S} f(x) = 1$$

Ejemplo: Lanzamiento de un dado.

Distribución de Probabilidad | Distribución Continua

El resultado de evaluar la función densidad en un punto del soporte NO es igual a la probabilidad de que la variable tome ese valor, de hecho, la probabilidad de que la variable tome un valor individual típicamente es cero.

La probabilidad de que la variable se encuentre en un rango de valores se evalua con una integral.

$$\int_a^b f(x)dx = Pr(a <= X <= b)$$

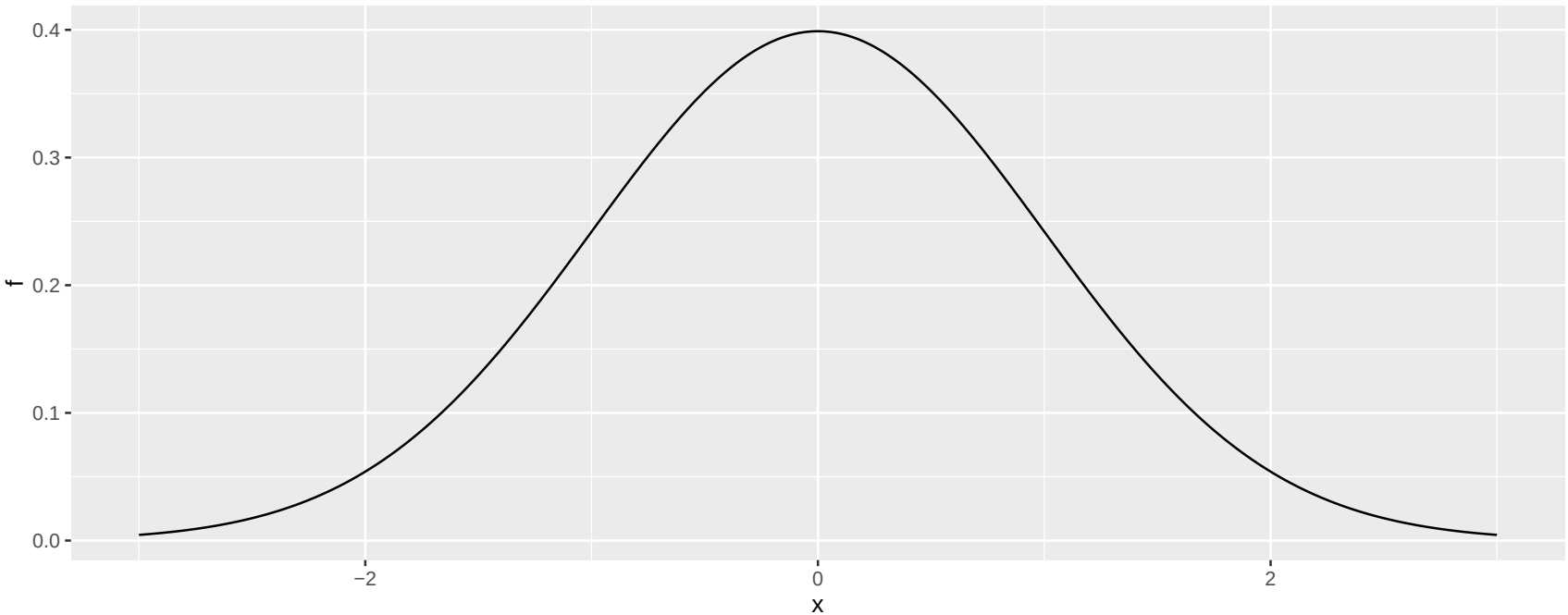
Además, la integral sobre todo el soporte debe ser igual a 1.

$$\int_{x \in S} f(x)dx = 1$$

Ejemplo: Altura de una persona.

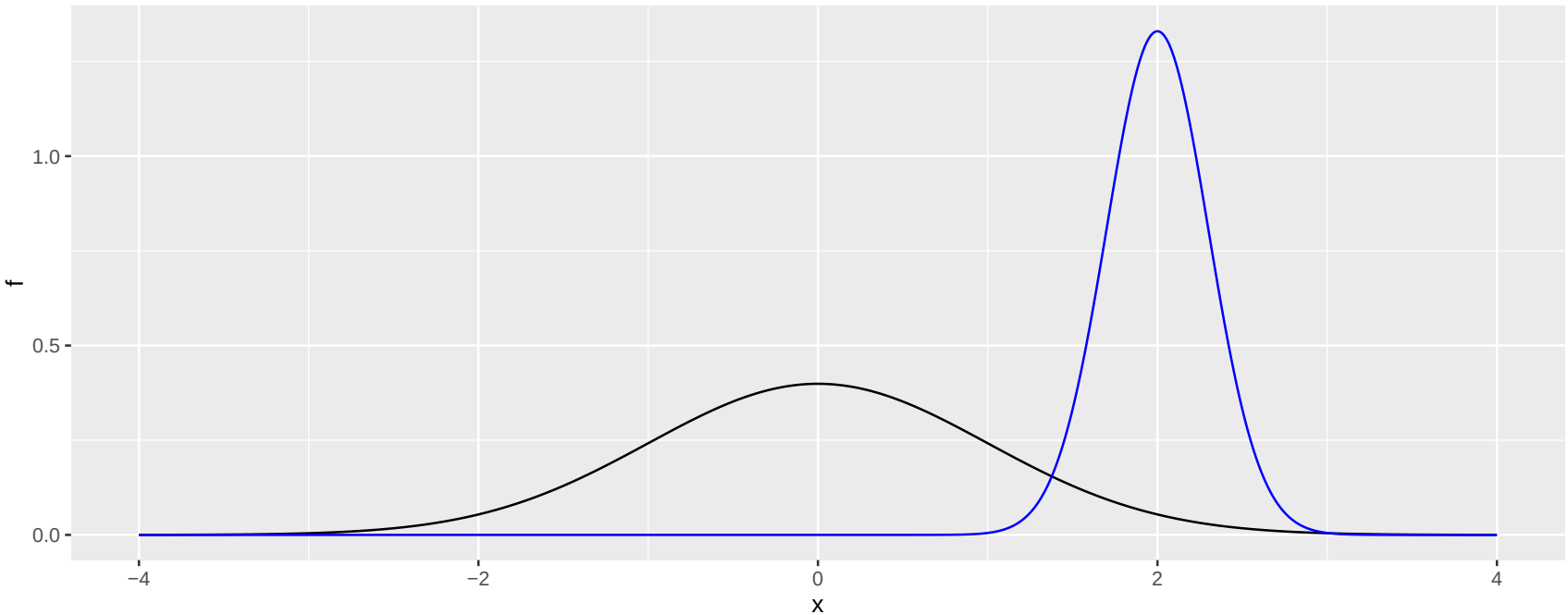
La distribución normal

La distribución normal es muy importante en la teoría estadística, y es muy prevalente en las aplicaciones. Una gran cantidad de fenómenos nat-
urales, al ser medidos, exhiben valores con un comportamiento normal.



La distribución normal

La distribución normal se caracteriza por su media (mu) y su varianza (sigma^2). La media es el parámetro de ubicación, nos dice en dónde se concentra la densidad. La varianza es un parámetro de precisión, nos dice que tan concentrada está la densidad



La distribución normal | Propiedades

- Es simétrica alrededor de la media (mu)
- La media es igual a la mediana y a la moda
- Cerca del 68% de la distribución se acumula en una desviación estándar alrededor de la media
- Cerca del 95% de la distribución se acumula en dos desviaciónes estándar alrededor de la media

Probabilidad Condicional

La probabilidad de que suceda un evento, dado algún otro, se conoce como probabilidad condicional. Se define como

$$Pr(A|B) = Pr(A \cap B)/Pr(B)$$

Aplicando esta definición podemos obtener el teorema de Bayes.

$$Pr(A|B) = Pr(B|A)P(A)/Pr(B)$$

Si $Pr(A|B) = Pr(A)$ se dice que A y B son independientes.

Probabilidad Conjunta

Cuando se tienen dos variables aleatorias X y Y, se puede pensar en una distribución de probabilidad conjunta, es decir, una ley que describa y asigne probabilidades a los posibles valores de ambas variables. En el caso discreto

$$Pr(X = x, Y = y) = Pr(X = x|Y = y)Pr(Y = y)$$

Densidad Conjunta

De las definiciones anteriores, se puede obtener que en el caso de dos variables aleatorias continuas X y Y, su función de densidad sigue las mismas reglas

$$f_{XY}(x, y) = f_{X|Y}(X|Y)f_Y(Y)$$

f_{XY} se conoce como *función de densidad conjunta*, y $f_{X|Y}$ es la *función de densidad condicional*, es decir, la ley que describe la probabilidad de los eventos en X dado un valor de Y.

Ambos casos se pueden combinar siguiendo las mismas reglas.

Valor esperado

El valor esperado de una distribución es el *promedio ponderado* de su función (masa ó de densidad) con pesos iguales al valor de la variable aleatoria. Se denota por $E[X]$, donde X es la variable aleatoria en cuestión.

En el caso discreto

$$E[X] = \sum_{x \in S} x f(x)$$

En el caso continuo

$$E[X] = \int_{x \in S} x f(x) dx$$

Ley de los grandes números

Dada una secuencia $X_1, X_2, ..., X_n$ variables aleatorias independientes e idénticamente distribuidas (i.i.d.), con valor esperado $E[X_1] = E[X_2] = ... = \mu$, el promedio muestral converge al valor de μ cuando n tiende a infinito

$$\frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{n\to\infty} \mu$$

Esta ley garantiza un comportamiento estable en fenómenos o experimentos repetidos una gran cantidad de veces, y asegura que el promedio muestral en una secuencia de valores que se cree son i.i.d. sea un buen estimado del valor esperado de la distribución

Ley de los grandes números | Borel

Si se tiene un experimento que se repite una gran cantidad de veces, la proporción de veces que un evento dado sucede en dicho experimento se acerca a la probabilidad de que el evento ocurra. Si E es el evento, p la probabilidad de que ocurra y $N_n(E)$ el número de veces que ocurre en n experimentos:

$$\frac{N_n(E)}{n} \xrightarrow{n\to\infty} p$$

Esto quiere decir que no sólo el promedio se acerca al valor esperado, sino que todos los posibles eventos adquieren la proporción descrita por la ley de probabilidad.

Teorema Central de Límite

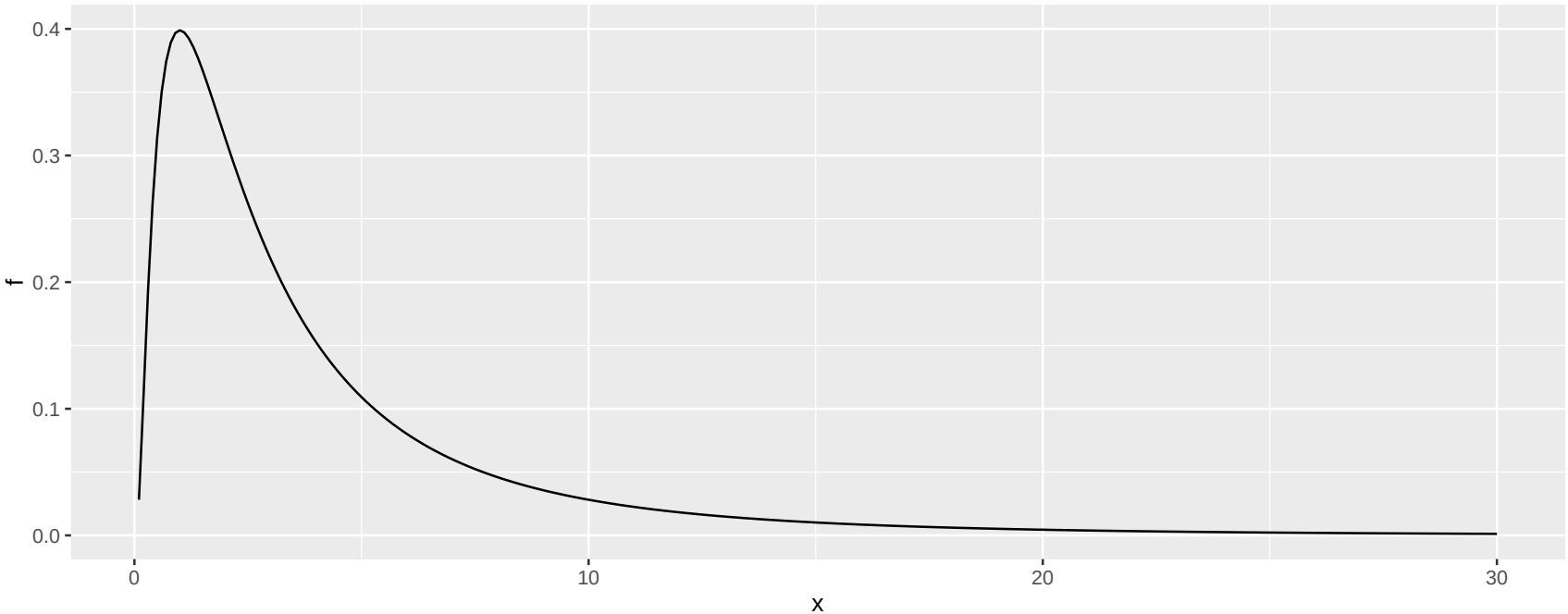
Dada una secuencia de variables aleatorias i.i.d, sabemos que su promedio muestral converge a su valor esperado. Además, si tienen varianza finita σ^2, el promedio muestral converge en distribución a una normal con media μ y varianza σ^2/n, sin importar la distribución original.

$$\frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{d} N(\mu, \sigma^2/n)$$

Este teorema coloca a la distribución normal en una posición central en la teoría estadística y explica su prevalencia en fenómenos naturales donde una medición es la suma de muchos pequeños efectos.

Ejemplos de distribuciones | Log Normal

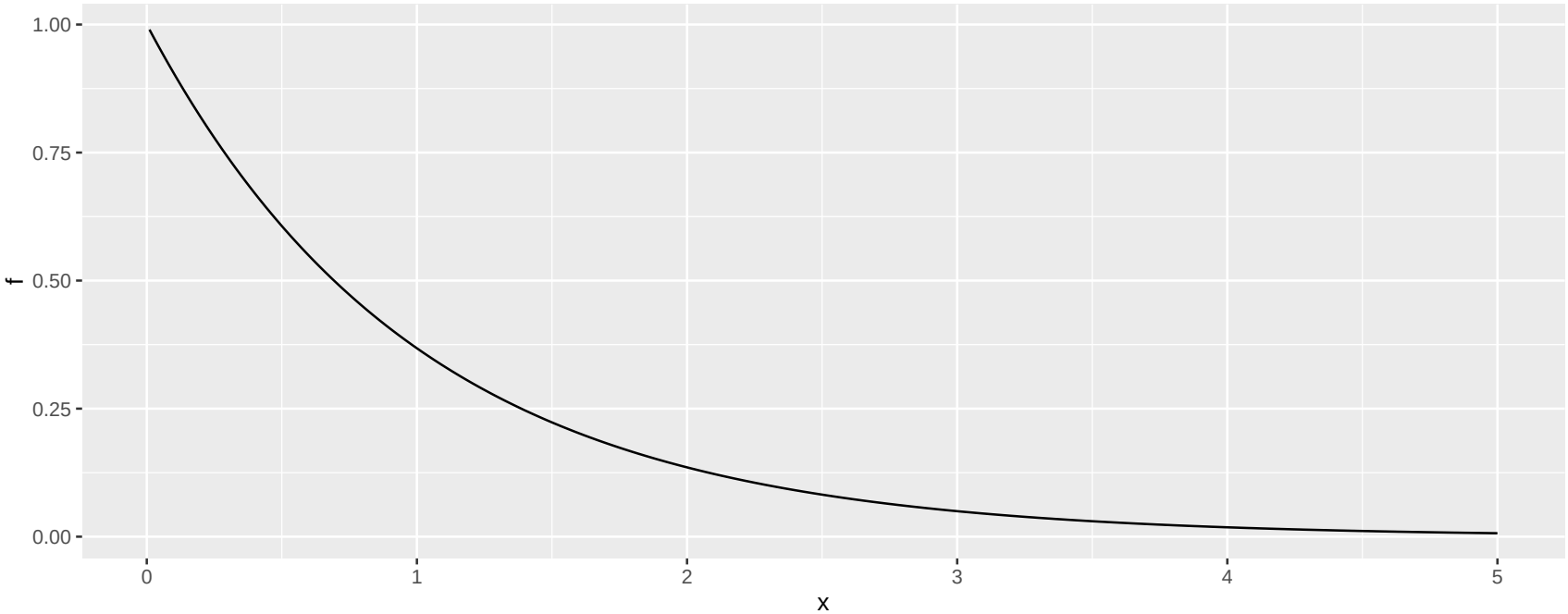
Decimos que X sigue una distribución log normal cuando $Y = log(X)$ sigue una distribución normal.



$\mu = 0, \sigma = 1$

Ejemplos de distribuciones | Exponencial

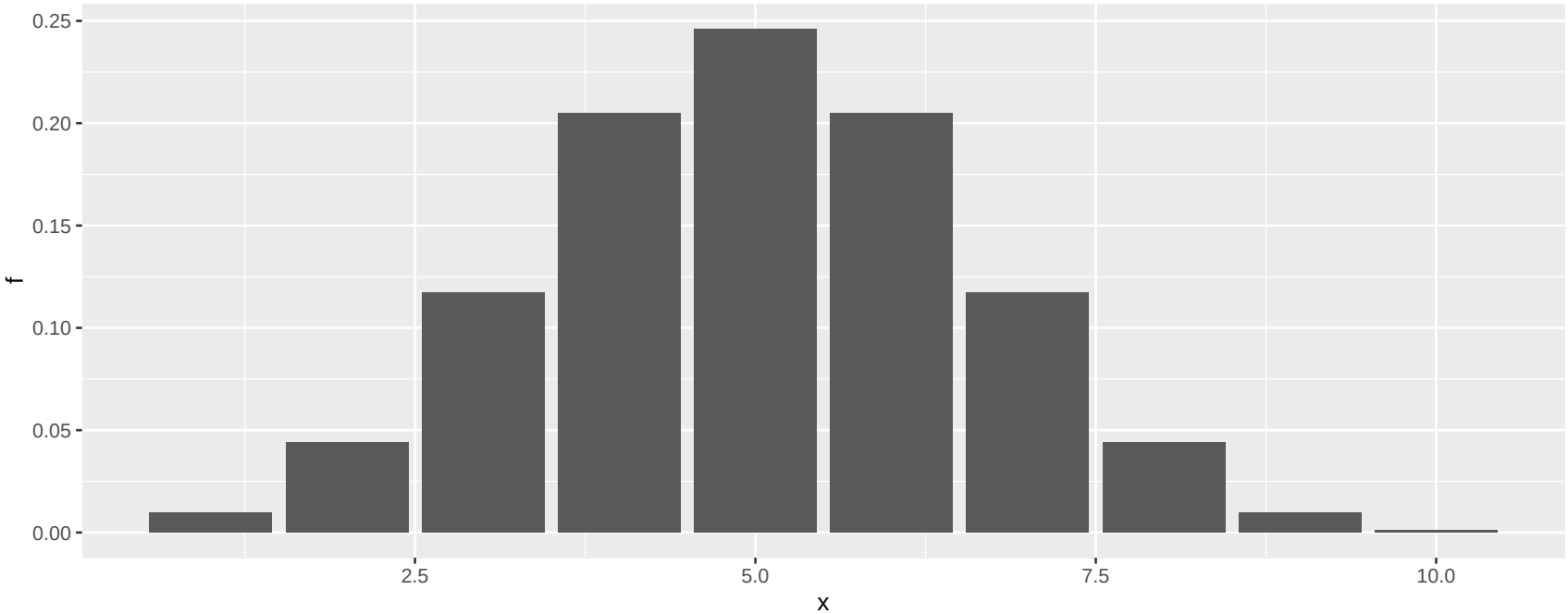
La distribución exponencial se puede utilizar para modelar tiempos de espera entre eventos independientes, como tiempo en una fila, o tiempo de llegada de clientes.



$\lambda = 1$. $1/lambda$ indica el tiempo promedio de llegada

Ejemplos de distribuciones | Binomial

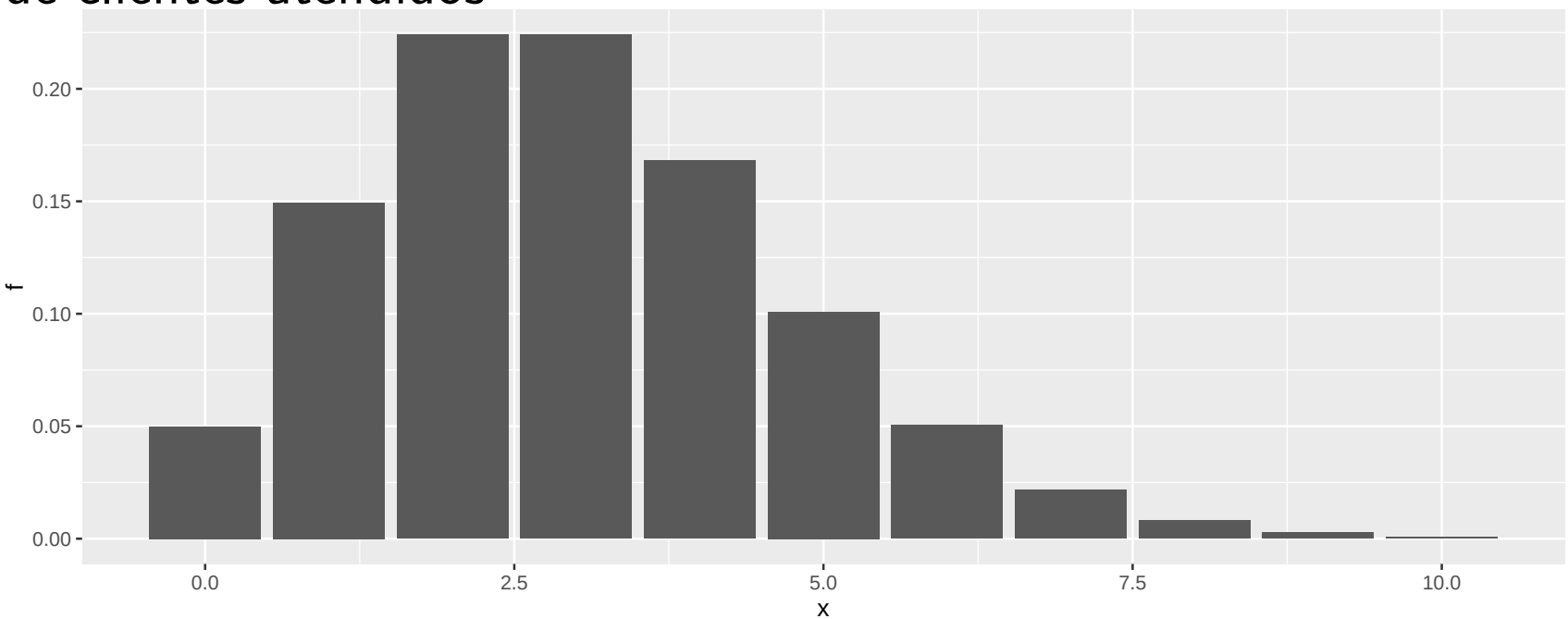
La distribución binomial se puede utilizar para modelar el número de éxitos en algún experimento después de n intentos, con probablidad de éxito p



$n = 10, p = 0.5$

Ejemplos de distribuciones | Poisson

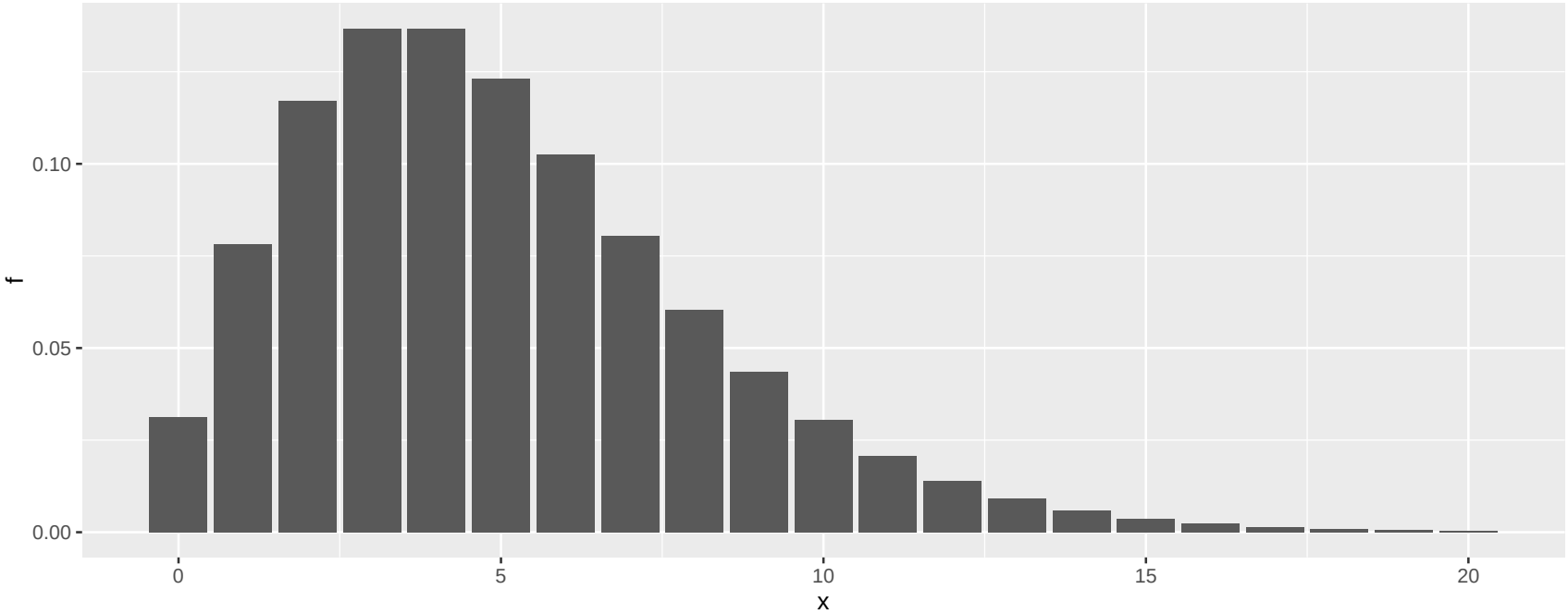
La distribución poisson se utiliza para modelar eventos independientes que suceden con tiempo de espera exponencial, o a una misma tasa, como número de peticiones a un servicio o número de clientes atendidos



$\lambda = 3$ Una propiedad importante es que $\lambda = E[X] = Var(X)$.

Ejemplos de distribuciones | Binomial Negativa

La distribución binomial negativa se puede utilizar para modelar el número de experimentos fallidos hasta alcanzar n éxitos, con probabilidad de éxito p



$n = 5, p = 0.5$

También puede ayudar a modelar cantidades positivas discretas dónde la varianza es más grande que la media, como eventos contagiosos o número de acciones en una plataforma.

Estimación de parámetros

Supongamos que tenemos una muestra aleatoria $X_1, X_2, ..., X_n$, es decir, una secuencia de variables aleatorias i.i.d. y sabemos que fueron tomadas de una distribución f con parámetros desconocidos θ.

La densidad conjunta de está muestra está dada por

$$f(x_1, x_2, ..., x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

Una manera de estimar el valor de los parámetros theta es maximizar esta función con respecto a θ, este método se conoce como *máxima verosimilitud*. A la función $L(\theta; x_1, ..., x_n) = f(x_1, ..., x_n|\theta)$ se le conoce como función de verosimilitud

Estimación de parámetros

Típicamente, no se maximiza directamente la función de verosimilitud, sino su logaritmo, pues el resultado es el mismo pero es mucho más conveniente trabajar con sumas que con productos.

$$logL(\theta; x_1, ..., x_n) = \sum_{i=1}^{n} logf(x_i|\theta)$$

Para la mayoría de las distribuciones populares se conoce una forma cerrada de este estimador, por lo que basta buscarla en algún material confiable.

Estimación de parámetros

El paquete MASS implementa una función que hace muy fácil el cálculo de estas estimaciones. Incluye por nombre muchas distribuciones populares, y permite el cálculo de cualquier densidad si se usa una función.

```
x <- c(8, 11, 14, 13, 9, 9, 10, 12, 8)
MASS::fitdistr(x, densfun = "normal")
```

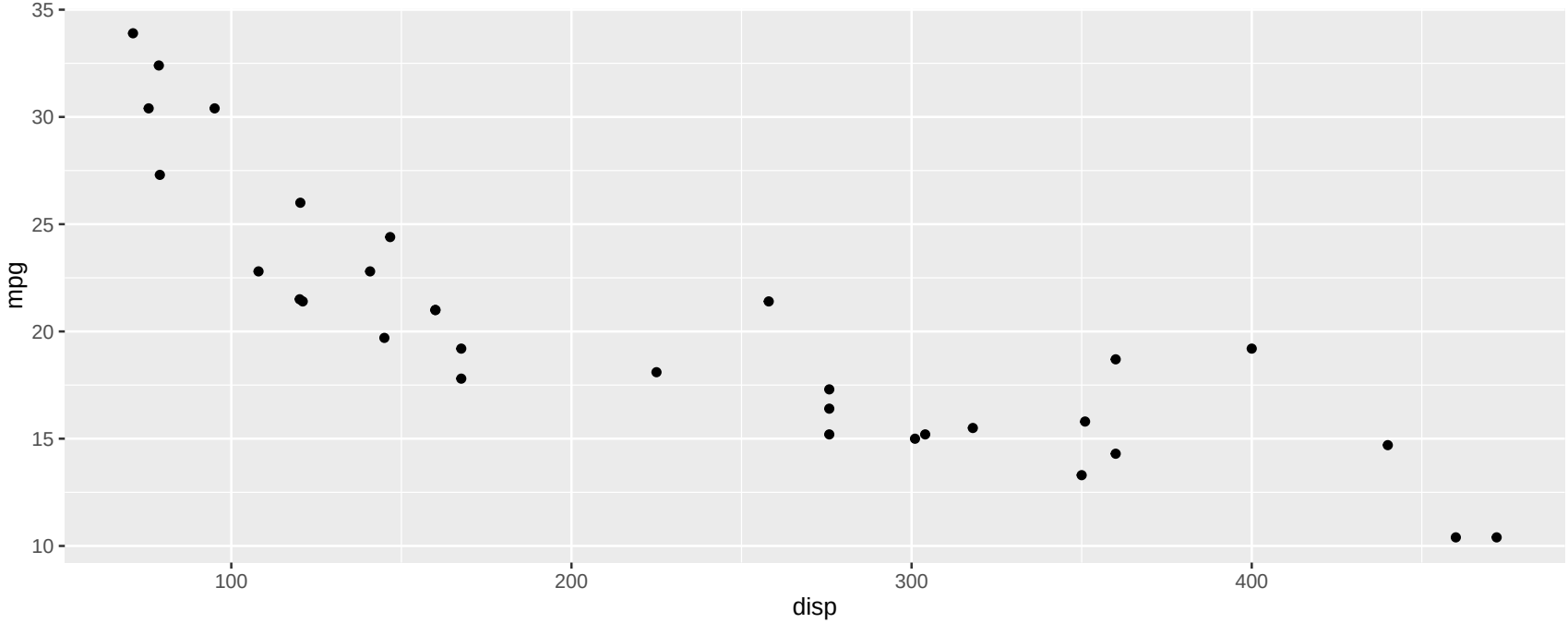
```
##        mean          sd
##    10.4444444    2.0608041
##   ( 0.6869347) ( 0.4857362)
```

```
MASS::fitdistr(x, dnorm, list(mean=10, sd=2))
```

```
##        mean          sd
##    10.4444364    2.0608051
##   ( 0.6869350) ( 0.4857362)
```

Modelo lineal

Consideremos una gráfica de la base de datos de automóviles



Es claro que hay una relación entre ambas variables. ¿Cómo podemos encontrarla y cuantificarla?

Modelo lineal

El modelo lineal supone que la relación entre X y Y es

$$Y_i = \alpha + \beta X_i + \epsilon_i$$

donde $\epsilon_i \sim N(0, \sigma_\epsilon^2)$. Este término modela el "error" o variabilidad alrededor de la forma funcional. Estos errores se suponen independientes de X y entre sí.

Si consideramos los valores de X fijos, tenemos que

$$Y_i \sim N(\alpha + \beta X_i, \sigma_\epsilon^2)$$

Nuestro objetivo es encontrar las mejores estimaciones de α y β

Modelo lineal

Aplicando el método de máxima verosimilitud tenemos que, sin importar la varianza

$$\beta = \frac{cov(X, Y)}{var(X)}$$
$$\alpha = \bar{Y} - \bar{X}\beta$$

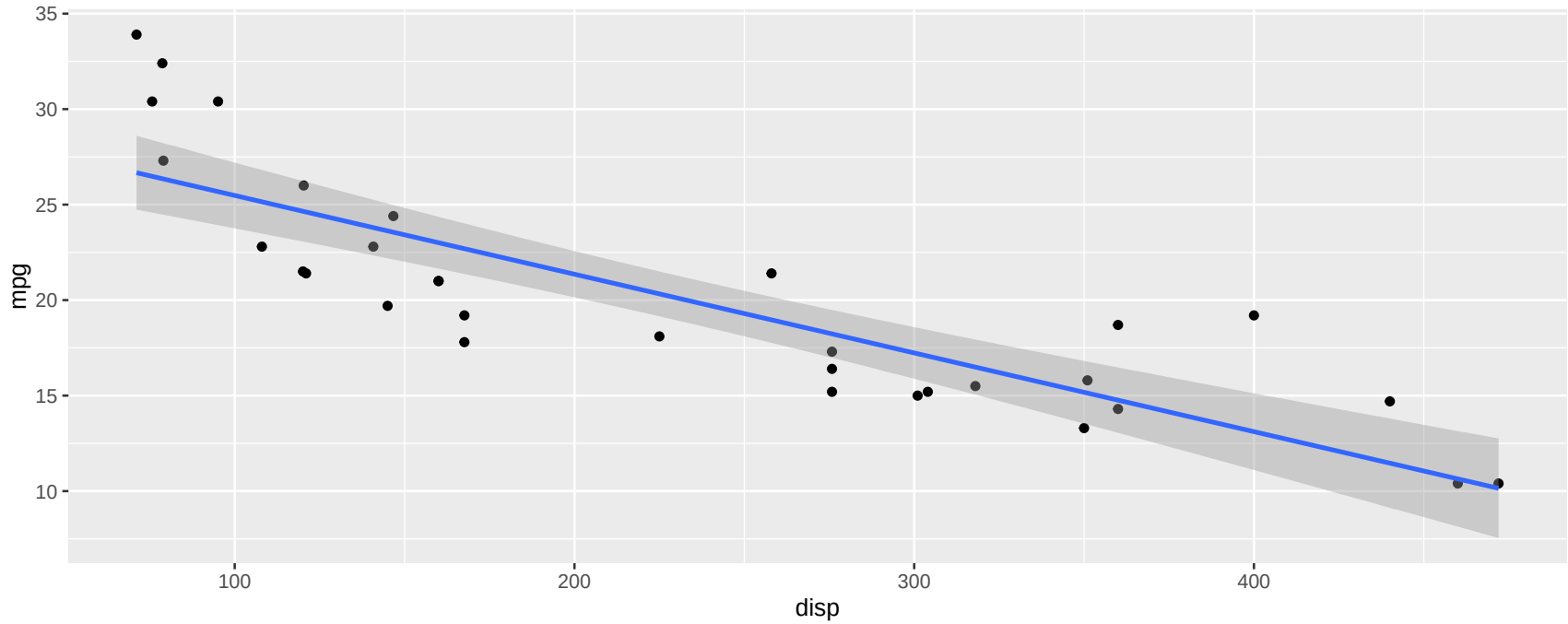
donde

$$cov(X, Y) = \frac{1}{n}\sum X_i Y_i - \bar{X}\bar{Y}$$
$$var(X) = \frac{1}{n}\sum X_i^2 - \bar{X}^2$$

y $\bar{X}$, $\bar{y}$ denotan el promedio muestral. Estos estimados coinciden con los de mínimos cuadrados.

Modelo lineal

Además, usando los cuantiles de la distribución de Y_i, podemos encontrar un intervalo de "confianza", alrededor del cual esperamos que se encuentre la media de la distribución.



Modelo lineal

La manera más sencilla de hacer esto en R es utilizar la función `lm`

```
summary(lm(mpg~disp, mtc))
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtc)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8922 -2.2022 -0.9631  1.6272  7.2305
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 29.599855   1.229720  24.070  < 2e-16 ***
## disp        -0.041215   0.004712  -8.747 9.38e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
##
## Residual standard error: 3.251 on 30 degrees of freedom
## Multiple R-squared:  0.7183,  Adjusted R-squared:  0.709
```

Valor p

Dadas dos hipótesis opuestas acerca de un conjunto de datos, una de las cuales se busca rechazar (la hipótesis nula), un valor p indica la probabilidad de obtener un conjunto de datos más "extremo" que lo que se observó, si la hipótesis nula fuera verdad. Intuitivamente, un valor p pequeño indica que la hipótesis nula debe ser rechazada.

En el caso del modelo lineal, la hipótesis nula indica que el coeficiente de pendiente sea igual a cero, por lo que un valor pequeño está relacionado con un buen ajuste.

R^2

El coeficiente de determinación, o R cuadrada, es otra medida del ajuste de un modelo. Dadas n observaciones y_i y n predicciones f_i, el coeficiente se define como

$$R^2 = 1 - \frac{\sum(y_i - f_i)^2}{\sum(y_i - \bar{y})^2}$$

Valores cercanos a 1 indican un buen ajuste, pues indican que las predicciones f_i son cercanas a los valores y_i, relativo a la varianza en dichos valores.

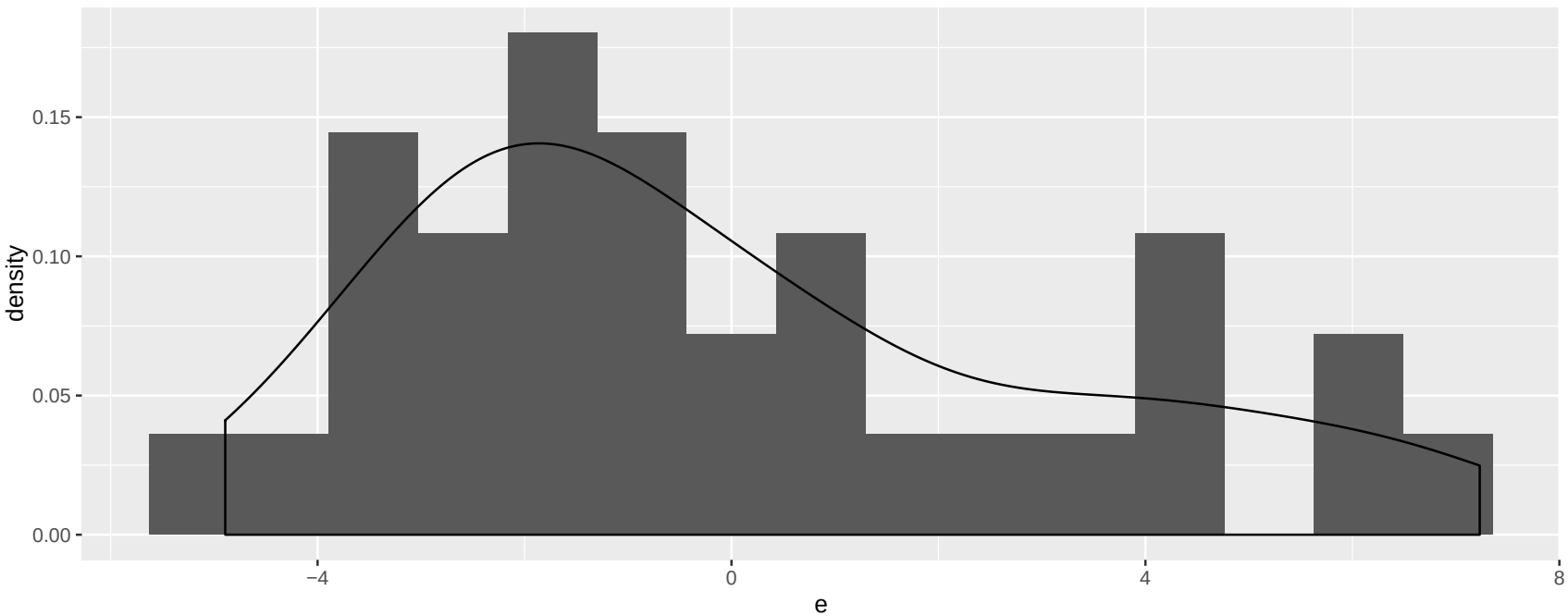
Insuficiencia de p y R^2

Nuestro modelo lineal exhibe un valor p muy pequeño y una R^2 de 0.7, típicamente considerada buena, ¿podemos decir que es un buen modelo?

Insuficiencia de p y R^2

Los residuales no se comportan de manera normal (Supuesto de error normal)

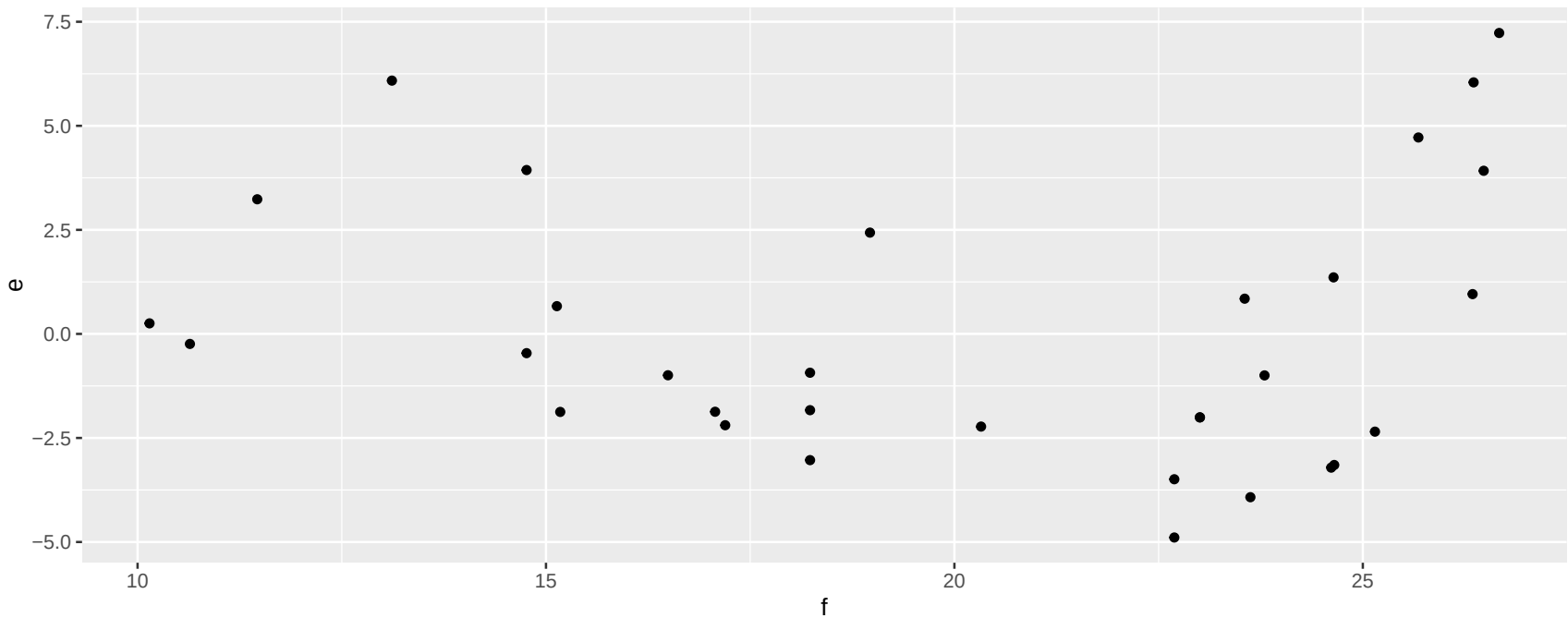
```
model <- lm(mpg~disp, data=mtc)
ggplot(data.frame(e=model$residuals), aes(x=e, y=..density..)) +
  geom_histogram(bins=15) + geom_density()
```



Insuficiencia de p y R^2

La variabilidad (spread) de los residuales no se mantiene a través de los niveles de la variable de respuesta (Supuesto de varianza constante)

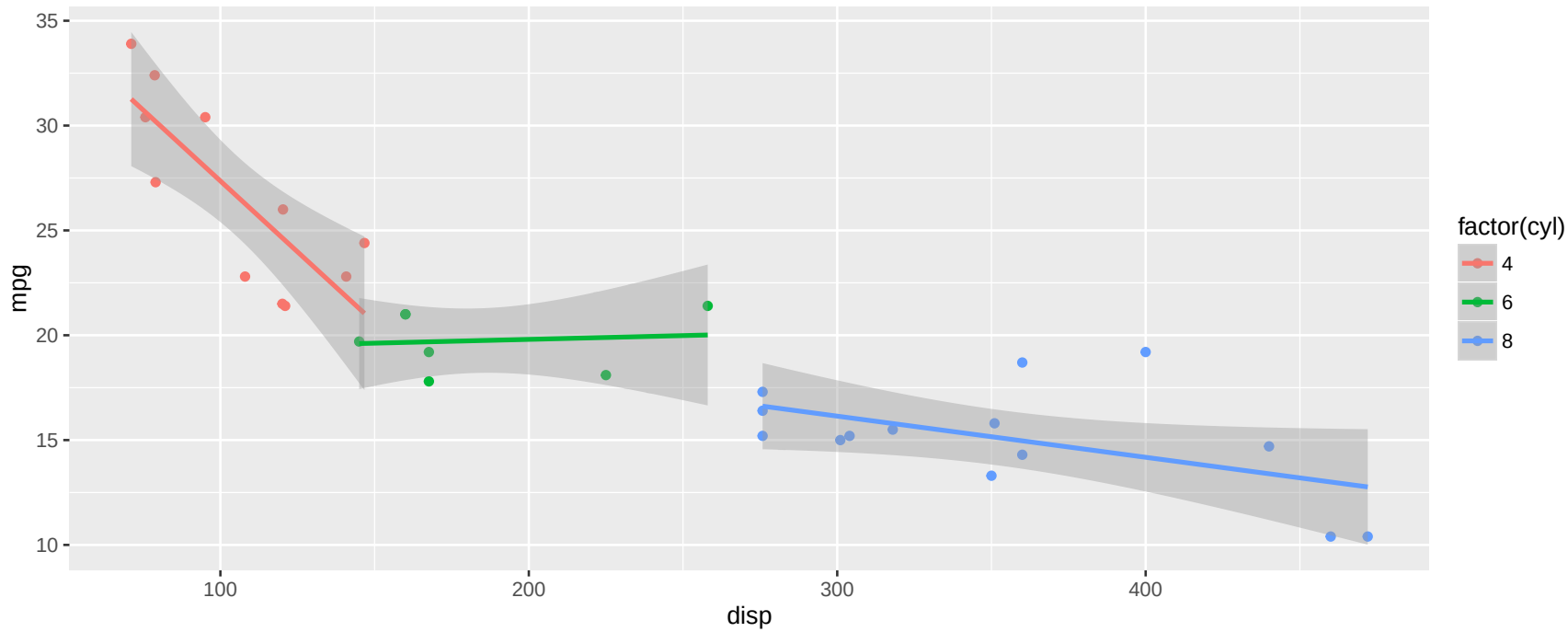
```
ggplot(data.frame(e=model$residuals, f=model$fitted.values), aes
  geom_point()
```



Insuficiencia de p y R^2

De hecho, hubo un factor importante que no tomamos en cuenta

```
ggplot(mtc, aes(x=disp, y=mpg, color=factor(cyl))) +
  geom_point() + geom_smooth(method="lm")
```



Insuficiencia de p y R^2

En resumen, el valor p y la R^2 no son suficientes para evaluar el buen ajuste de un modelo. Siempre es conveniente comprobar los supuestos, y, sobre todo, la habilidad de predicción de un modelo cuando se le alimentan datos nuevos, es decir, su capacidad para generalizar el fenómeno.

Interpretaciones de la teoría | Frecuentista

La probabilidad indica una medida objetiva, la probabilidad de un evento es igual a la frecuencia que se alcanza conforme el número de repeticiones del experimento crece.

No se puede hablar de la probabilidad de un valor fijo (como un parámetro) o de eventos ya sucedidos. Así mismo, es imposible hablar de la probabilidad de eventos que por su naturaleza provienen de un experimento que se repite una sola vez.(e.g. terremotos)

Interpretaciones de la teoría | Bayesiana o Subjetiva

La probabilidad indica una medida subjetiva, la probabilidad de un evento es la medida de incertidumbre o grado de creencia que un individuo tiene acerca de un evento.

Se puede hablar de la probabilidad de un valor fijo o de eventos sucedidos, así como de eventos únicos. Pero es necesario establecer de antemano, a través de una distribución *a priori*, la creencia que tiene el individuo acerca de tal evento. Los resultados de un análisis podrían no ser consistentes con distintas distribuciones *a priori*, aunque en el límite (con suficientes datos) sí lo son.

Data Science vs Statistics

La teoría estadística, junto con todos los métodos que describe, forman un conjunto de herramientas extremadamente poderoso, pero fácil de utilizar de manera errónea. Un entendimiento y sensibilidad tanto de la teoría como de sus aplicaciones es necesaria. La teoría estadística es esencial para el científico de datos.

Por otro lado, dominar perfectamente la teoría sin saber como explotarla o las herramientas computacionales necesarias para ello, paralizan al científico de datos, que siempre requeriría de ayuda de ingenieros para poder desempeñar.

Aprendizaje Estadístico

- Conclusiones de Teoría de la Decisión
- El problema de aprendizaje supervisado
- Error = Sesgo + Varianza. Tradeoff
- Overfitting
- La maldición de la dimensionalidad
- KISS: Model Edition
- Medición del error (sesgo + varianza)
- Cross Validation. Bootstrap. Out of bag
- Regresión. Métricas de error
- Clasificación. Métricas de error
- Modelos lineales generalizados
- Lasso, ridge, glmnet
- Árboles de decisión. Random Forests

Conclusiones de Teoría de la Decisión

El teorema de utilidad de von Neumann establece que, bajo ciertos supuestos de "racionalidad", y con opciones inciertas (probabilísticas), un agente buscará maximizar su utilidad esperada $E[U(X)]$, donde U representa la función de utilidad y X la variable aleatoria describiendo los posibles resultados.

Las hipótesis del teorema establecen reglas acerca de las preferencias del agente, e.g. que si A es preferido a B y B es preferido a C, A es preferido a C (transitividad). Estas son usadas en economía y en otros campos y forman la base de la teoría estadística bayesiana.

Sea $L(X) = -U(X)$ la función de pérdida. Entonces minimizar la périda esperada $E[L(X)]$ es equivalente a lo anterior.

Una nota importante es que la teoría de la decisión es *prescriptiva*, no *descriptiva*, es decir, indica cómo deberían comportarse los agentes, no describe como lo hacen en realidad.

El problema de aprendizaje supervisado

Supongamos que tenemos variables de entrada $X \in \mathbb{R}^p$, y una variable de salida $Y \in \mathbb{R}$. Buscamos una función $\hat{f}(X)$ para estimar el valor de Y.

Para poder escoger una función, necesitamos establecer una *función de perdida* $L(Y, \hat{f}(X))$ que penalize los errores de f. Una elección conveniente y popular es el error cuadrático $L(x, y) = (x - y)^2$. Entonces buscamos minimizar

$$E[L] = E[(Y - \hat{f}(X))^2]$$

Condicionando en X:

$$E[L] = E_x[E_{y|x}[(Y - \hat{f}(X))^2|X)]]$$

Por lo que basta minimizar f punto por punto. La solución a este problema es $\hat{f}(x) = E[Y|X = x]$

El problema de aprendizaje supervisado

El problema de aprendizaje entonces es encontrar o estimar la *esperanza condicional* de la variable de respuesta, dados los valores de las variables de entrada.

El modelo lineal hace esto suponiendo una distribución para $Y|X = x$, explícitamente

$$Y|X = x \sim N(\alpha + \beta^T X, \sigma^2)$$

Y estimando los parámetros α y β.

El problema de aprendizaje supervisado

Otra manera sencilla de estimar la esperanza condicional es directamente, con un promedio local. Sea x un punto fijo, y N_x una vecindad alrededor del punto. Entonces

$$\hat{y} = \frac{1}{n(N_x)} \sum_{p \in N_x} p$$

Es una estimación de la esperanza condicional. Este modelo se conoce como vecinos cercanos. Cuando en lugar de escoger una vecindad N_x se escogen los k puntos más cercanos a x, estamos hablando de k-vecinos cercanos.

Error = Sesgo + Varianza

Consideremos el error de predicción esperado $E[(Y - \hat{f}(X)^2)]$, donde $\hat{f}$ es una función que depende de X y Y, si consideramos los valores de X como fijos:

$$E[(Y - \hat{f}(X)^2)] = E[(Y - \hat{f}(X))^2 | X = x_0]$$

Supongamos que $Y = f(X) + \epsilon$ donde $f(X)$ es la "verdadera" función y $E[\epsilon] = 0$, $Var(\epsilon) = \sigma^2$. Entonces podemos probar que:

$$E[(Y - \hat{f}(X))^2 | X = x_0] = \sigma^2 + Sesgo(\hat{f}(x_0))^2 + Var(\hat{f}(x_0)^2)$$

Donde $Sesgo(\hat{f}) = (f(X) - E[\hat{f}(X)])$.

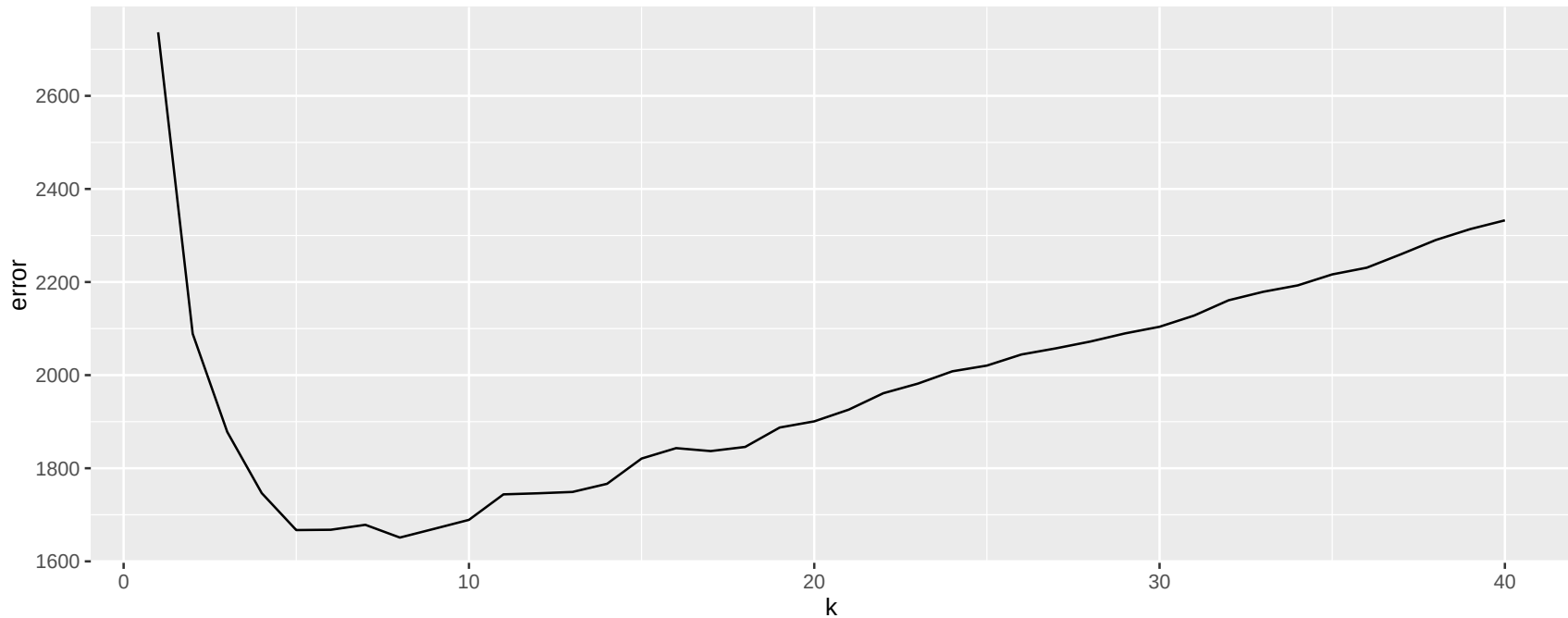
Error = Sesgo + Varianza

El primer término es el error inherente al fenómeno, es un *error irreducible*, los otros dos están bajo nuestro control. El sesgo indica la tendencia a subestimar o sobreestimar consistentemente, mientras que la varianza representa el spread alrededor de la predicción.

El siguiente experimento muestra como interactuan estos dos términos al ajustar un modelo de k-vecinos para diferentes valores de k

Error = Sesgo + Varianza

El parámetro k representa la complejidad del modelo, el modelo es más complejo cuando $k = 1$, y es el modelo más sencillo posible (constante), cuando $k = n$. Conforme k sube, el modelo se estabiliza, la varianza baja, pero el sesgo sube.



Overfitting

En el experimento anterior, cuando $k = 1$, el modelo está buscando el punto más parecido a x en el conjunto de entrenamiento, esto implica que ligeras variaciones en ese conjunto resultan en variaciones en las predicciones (alta varianza), además, si el punto más cercano a x no es suficientemente similar, o el valor en el conjunto de entrenamiento es un outlier, esto resulta en un error más grande.

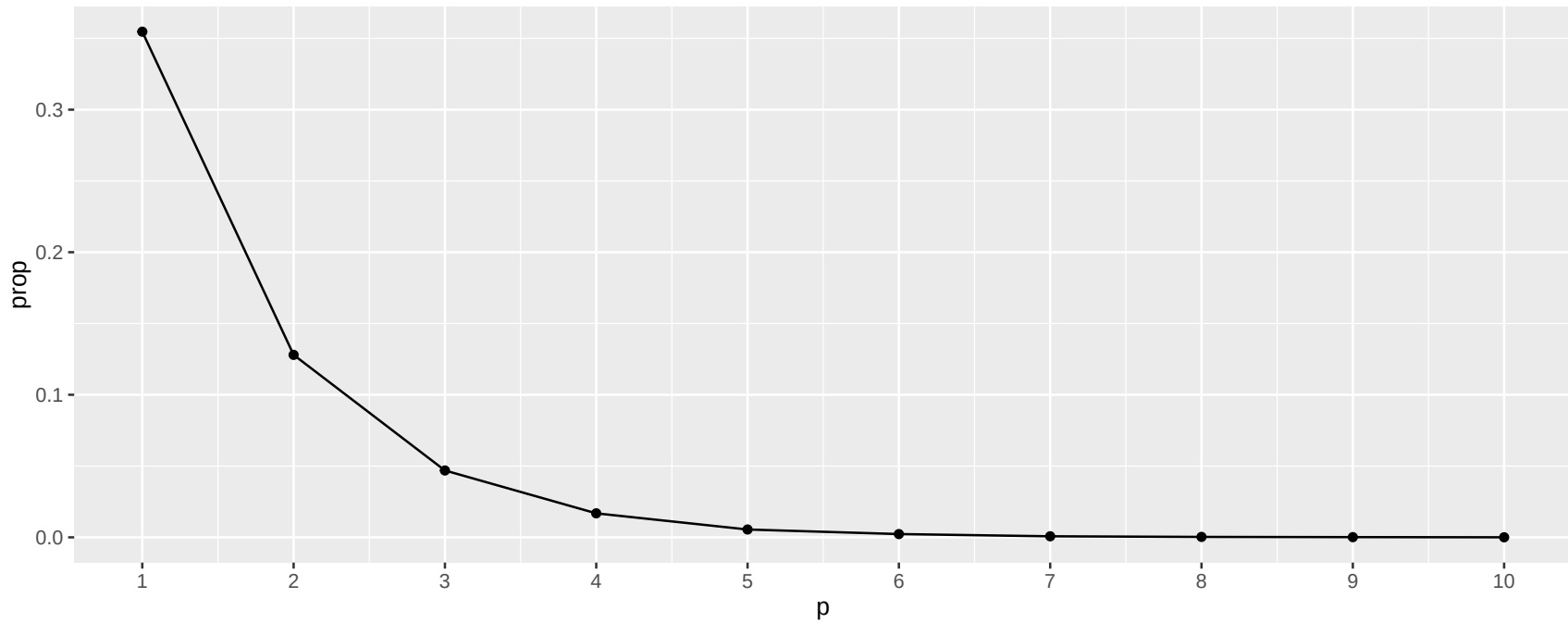
Este fenómeno se conoce como *overfitting*, o *sobreajuste*, que indica que nuestro modelo sigue demasiado al conjunto de entrenamiento, y perdió la habilidad de generalizar el fenómeno. Los errores por overfitting son típicamente mucho más grandes que los de un modelo simple, por lo que deben de evitarse usar modelos complejos en medida de lo posible.

El overfitting se puede diagnosticar comparando el error de entrenamiento contra el error de predicción. Si el primero es muy chico aunque el segundo no, podemos pensar que existe sobreajuste en nuestro modelo.

La maldición de la dimensionalidad

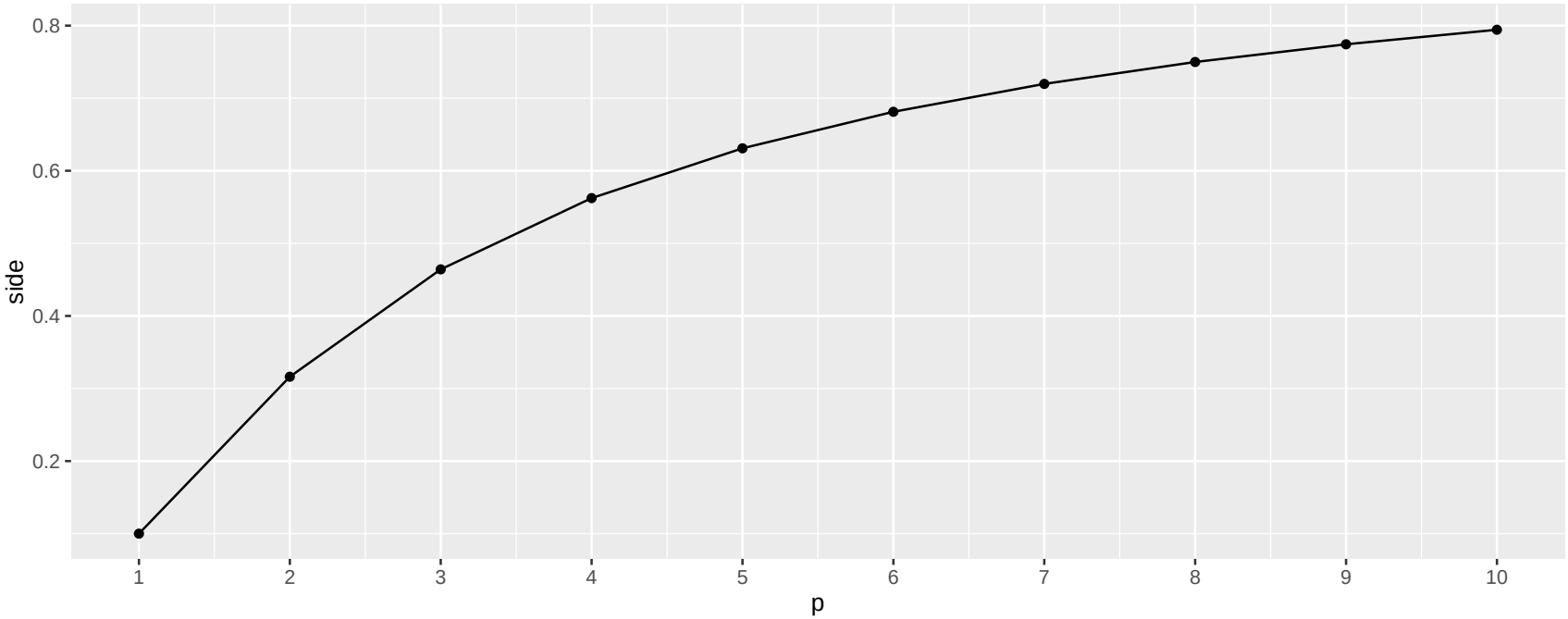
Consideremos el siguiente experimento. Sea $U_1, ..., U_n$ una muestra aleatoria uniforme en el hipercubo unitario en $\mathbb{R}^p$, es decir, todas las componentes de U_i se distribuyen uniforme en el intervalo $[0, 1]$

Tomemos un punto x al azar y un hipercubo alrededor de tamaño r con x como punto medio. Entonces la proporción esperada de U_i que capturaremos será r^p. La gráfica ilustra este resultado para $r = 0.4$ con una simulación.



La maldición de la dimensionalidad

De manera equivalente, el tamaño esperado del lado de un hipercubo que necesitamos para capturar la proporción r de los datos alrededor de un punto x, es de $r^{1/p}$. En otras palabras, si tenemos p dimensiones y un punto x, necesitaremos cubrir $r^{1/p}$ del espacio para capturar una proporción r de los datos. La gráfica representa esto para $r = 0.1$



Lo que se intenta mostrar es que, en dimensiones grandes, es difícil encontrar puntos "parecidos" a un punto arbitrario en nuestro

KISS

La conclusión de los dos temas anteriores, es que los modelos deben de mantenerse “simples” en medida de lo posible. Introducir mucha complejidad como en el caso de k-vecinos con k muy pequeña lleva un riesgo de sobreajuste, mientras que introducir una gran cantidad de variables nos puede llevar a caer en la “maldición” de la dimensionalidad.

Reducción de dimensionalidad

Existen varias técnicas para evitar los problemas que conlleva utilizar dimensiones muy grandes. Listo algunas de las más populares, sin orden en particular.

- PCA (Principal Component Analysis): Dado un conjunto de datos con valores reales. $X \in \mathbb{R}^{p \times n}$, PCA encuentra la transformación lineal tal que el primer componente tiene la varianza más alta posible, y cada componente siguiente la varianza más alta posible con la restricción de que sea ortogonal a los componentes anteriores. Es recomendable usar PCA cuando la varianza explicada por los primeros componentes es relativamente alta comparada con la varianza total del conjunto original. `prcomp` implementa este algoritmo en R.

Reducción de dimensionalidad

- FA (Factor Analysis): Tenemos p variables x_i con medias μ_i. Supongamos que $x_i - \mu_i = \sum_{j=1}^{n} l_i j F_j + \epsilon$. Si además $E[F_j] = 0$ y $Cov(F_i, F_j) = 0$, las soluciones F_j al sistema se llaman los "factores". Existen muchos métodos para encontrar esta solución. Una ventaja de FA es que puede ser utilizado con variables categóricas, a diferencia de PCA. `factanal` implementa este algoritmo en R, encontrando la solución a través del método de máxima verosimilitud.

Reducción de dimensionalidad

- MDS (Multi-Dimensional Scaling): Si tenemos un conjunto de observaciones X en donde podemos definir una métrica (distancia) d, definimos D como la matriz de distancias ($D_{ij} = d(X_i, X_j)$). El algoritmo de MDS busca encontrar las coordenadas en un espacio (euclidiano) de dimension N, de tal manera que las distancias se conserven de la mejor manera posible. Una ventaja de MDS es que no es necesario imponer ninguna restricción en los datos, basta que podamos definir una métrica entre dos observaciones. `cmdscale` implementa este método en R.

Medición del error

El error de entrenamiento tiende a ser optimista en cuanto al error de predicción, esto es porque una vez que encontramos datos fuera del conjunto de entrenamiento, el modelo necesita generalizar el fenómeno. Además, aunque el error de entrenamiento sea muy chico, corremos el riesgo de caer en sobreajuste. Necesitamos otro modo de estimar el error de predicción.

La mejor alternativa es dividir el conjunto de datos disponible en tres partes: entrenamiento, validación, y prueba. El primero es utilizado para entrenar todos los modelos, el segundo para realizar una selección del modelo, y el tercero como una última prueba, para estimar el error de generalización. Un split típico puede ser 60% entrenamiento, 20% validación, y 20% prueba. Esta partición se debe de realizar de manera aleatoria.

Medición del Error

Sin embargo, esto requiere tener una gran cantidad de datos, y en la mayoría de los casos, preferiríamos usar todos los datos disponibles para alimentar un modelo. Existen tres métodos populares para abilitar esto:

- Cross Validation
- Bootstrap
- Out of bag

Aunque se uso uno de estos métodos, es recomendable "apartar" una parte del conjunto de datos para realizar una última estimación del error de predicción.s

Cross Validation

Se toma el dataset y se divide en k partes, después, se repite lo siguiente para cada parte i:

- Se juntan todas las partes EXCEPTO i
- Se entrena el modelo en ese conjunto
- Se calcula el error de predicción en la parte i

Finalmente, el error es promediado. Este método nos da una manera de utilizar el propio conjunto de entrenamiento. El promedio y la varianza de estos errores nos otorgan un método un poco más robusto para estimar el error de predicción. El paquete `caret` y `cvTools` ofrecen funciones para realizar validación cruzada.

Bootstrap

La idea es similar a CV en cuanto a que se obtienen distintos errores de predicción de modelos entrenados en subconjuntos del dataset original. El método bootstrap está bien establecido en la teoría estadística como manera de estimar cualquier función de un conjunto de datos con una distribución arbitraria.

- Tomamos B muestras de tamaño R del conjunto de datos original, con reemplazo.
- Para cada muestra, se entrena el modelo usandola como conjunto de entrenamiento y se calcula el error de predicción en todos los puntos que NO están en la muestra.
- Se promedia el error de predicción en cada punto, y finalmente se calcula el error de predicción general.

La ventaja de bootstrap sobre CV es que se puede realizar una cantidad de veces arbitraria, y el resultado converge teóricamente al error de predicción "real". Además, nos permite evaluar el error en cada punto, no sólo en general. El paquete `boot` implementa funciones para hacer el resampling. El paquete `caret` incluye

Out of bag

Supongamos que ajustamos una serie de modelos T_i utilizando subconjuntos A_i de nuestro dataset X. Obtenemos un modelo final combinando las predicciones individuales de los modelos T_i, por ejemplo, promediando sus predicciones en cada punto. Este tipo de modelo se conoce como ensamble.

Entonces, una manera de estimar el error en un punto x es combinar las predicciones de todos los T_i que NO incluyen a x en su conjunto de entrenamiento y calcular el error entre x y dicha combinación. Las funciones que ajustan modelos de ensamble típicamente proveen una opción para estimar el error mientras se ajusta el modelo, por ejemplo `randomForest`.

Métricas de error (Regresión)

Hasta ahora sólo hemos hablado de manera abstracta de como estimar el error, sin proveer una métrica directa. En el caso de regresión, la manera más común, y que está atado a la elección de la esperanza condicional, es el error cuadrático medio. De hecho, por la ley de los grandes números:

$$\frac{1}{n}\sum_{i=1}^{n_t}(y_i - \hat{f}_i) \rightarrow E[L(Y, \hat{f}(x))^2]$$

Donde $\hat{f}_i$ es la estimación para el valor y_i en nuestro conjunto de prueba. Sin embargo, bajo ciertas condiciones, otras métricas pueden ser más adecuadas

Métricas de error (Regresión)

- Cuando existen outliers, que penalizan muchísimo errores en su estimación, se recomienda utilizar el error absoluto medio. otra opción es utilizar el error cuadrático mediano.
- Cuando la escala de la variable de respuesta es muy grande, de manera que valores altos de y pesan de manera más importante en el error, se recomienda utilizar el error cuadrático logarítmico. Esto es equivalente a realizar predicciones sobre $logy$, en lugar de y.
- Cuando existen casos que son más importantes para la aplicación del modelo, podemos pesar el error escogido. El peso de las observaciones que correspondan a dichos casos debe ser superior, y todos los pesos deben sumar a 1.

Métricas de error (Clasificación)

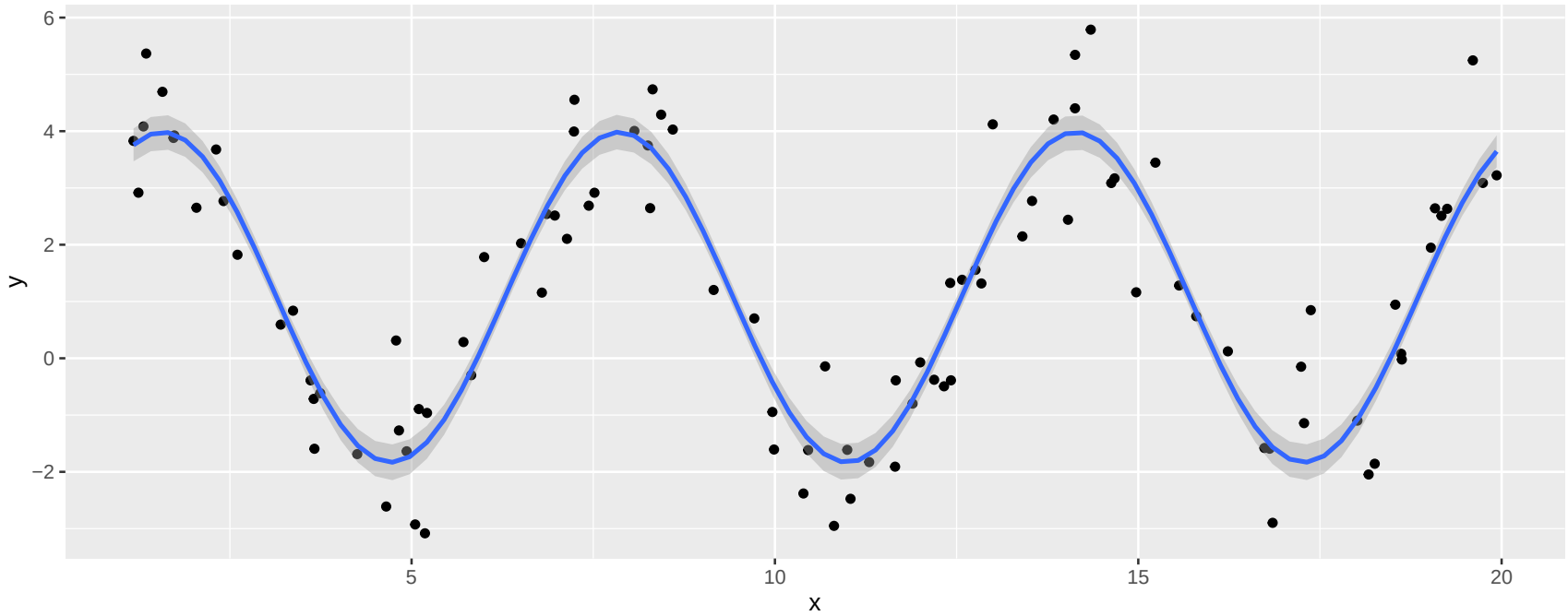
En el caso de clasificación, del que no se ha hablado mucho en estas notas, existen muchas más maneras de estimar el error, estas deben ser escogidas dependiendo del número de clases que existan en la variable a predecir, la importancia relativa de esos casos, entre otros factores. Se listan algunas de las más comunes, y se deja como ejercicio al lector averiguar su caso de uso e implementación.

- Proporción mal clasificada (para cada clase)
- Pérdida logarítmica
- Área debajo de la curva ROC
- Matriz de confusión

Modelos lineales generalizados

El modelo lineal no necesariamente implica una relación exclusivamente lineal entre X y Y, a continuación se ilustra un ejemplo.

```
data <- data_frame(x = runif(100, min=1, max=20), y = 3*sin(x)
ggplot(data, aes(x=x, y=y)) + geom_point() + geom_smooth(metho
```



Estos modelos también pueden partir de imponer otras distribuciones distintas a la normal en $Y|X$. Un caso popular es la regresión logística, donde se impone una distribución binomial a $Y|X$, con el fin de realizar

Regularización: Lasso, Ridge

GLM nos abre una gran cantidad de opciones en el modelado, a partir de la transformación de variables. Sin embargo, investigar si existe una relación con tantas posibles transformaciones es impráctico. Además, incluir gran cantidad de variables y transformaciones aumenta la complejidad del modelo, un efecto indeseable.

La regularización *penaliza* la complejidad en un modelo, restringiendo su complejidad, y en algunos casos, removiendo variables que aportan poco ajuste al modelo. Vemos dos casos de regularización para el modelo lineal.

Ridge

Recordemos que la estimación de parámetros en el modelo lineal consiste en resolver el problema

$$\hat{\beta} = argmin(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j))$$

La regresión ridge se puede definir añadiendo una penalización a este problema

$$\hat{\beta} = argmin(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j) + \lambda \sum_{j=1}^{p} \beta_j^2)$$

λ es el parámetro de regularización. El resultado es un modelo *menos* complejo que la regresión lineal, pues es naturalmente atraído hacia el modelo constante. En la optimización de ridge es prácticamente imposible que alguna de las β_j sea igual a cero, por lo que este modelo sólo debe utilizarse con variables informativas.

Lasso

De manera similar a ridge, lasso añade una penalización al problema lineal.

$$\hat{\beta} = argmin(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j) + \lambda \sum_{j=1}^{p} |\beta_j|)$$

A diferencia de ridge, la naturaleza de la restricción permite que algunas β_j se optimizen en cero, por lo que lasso puede ser utilizado para selección de variables. El modelo resultante es aún más rígido (menos complejo) que el que se obtiene con ridge

Elastic Net

Si se quiere evaluar un modelo con complejidad entre lasso y ridge, se puede añadir un tercer tipo de penalización, de la siguiente manera.

$$\hat{\beta} = argmin(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j) + (1-\alpha)\lambda \sum_{j=1}^{p} \beta_j^2 + \alpha\lambda \sum_{j=1}^{p} |\beta_j|)$$

Con $0 < \alpha < 1$. Este es exactamente el modelo que ofrece el paquete `glmnet`. La función además permite controlar la distribución que se impone a $Y|X$.

Árboles de decisión

Un árbol de decisión parte el espacio en el que se encuentra X, nuestro conjunto de datos, a través de una serie de reglas binarias, y ajusta un modelo local (típicamente constante) en cada región.

Intuitivamente, el objetivo del árbol es aislar puntos similares entre sí, y encontrar, para cada punto en donde se pida una predicción, la región a la que corresponde y un modelo local adecuado para ese punto. En ese sentido, es un intento de estimación directa de $E[Y|X]$

Decidir las reglas binarias y el nivel de complejidad de un árbol de decisión es un problema complejo, y todas las soluciones propuestas son aproximaciones numéricas. Típicamente, en cada paso, se busca la variable y el valor que minimizen el error cuadrático en cada una de las dos regiones resultantes.

Árboles de decisión

La profundidad del árbol es un parámetro que controla la complejidad del modelo. El árbol más complejo posible aisla cada punto, y es equivalente a k-vecinos con $k = 1$. El árbol más simple posible es equivalente al modelo constante.

La estrategia típica es crecer el árbol hasta que un tamaño mínimo de nodo es alcanzado, p.ej. 5 puntos, y después cortarlo según un parámetro de complejidad.

Random Forest

Recordemos el teorema central de límite.

$$\frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{d} N(\mu, \sigma^2/n)$$

Supongamos que tenemos n árboles T_i, construidos aleatoriamente, pero con las mismas reglas. Aplicando el teorema, el promedio del resultado de cada árbol para un punto x_0 en particular, es un estimado con el mismo sesgo, pero menos varianza que el de cada árbol individual. El mejor caso se da cuando cada árbol individual es insesgado, es decir $E[T_i(x_0)] = f(x_0)$ donde f es la función "real" que sigue Y con respecto a X ($Y = f(X) + \epsilon$).

Random Forest

¿Cómo construir los árboles de manera aleatoria, pero asegurando que son un predictor decente?

Para cada árbol b desde 1 hasta B (el número de árboles):

- Toma una muestra bootstrap Z de tamaño n_T del dataset X
- Crea un árbol de decisión T_b utilizando la muestra Z siguiendo los sig. pasos de manera recursiva hasta alcanzar nodos de n_m puntos:

1. Selecciona m variables de las p
2. Elige la mejor variable y split de entre las m
3. Divide el nodo en dos de acuerdo a este split

Para hacer una predicción en un punto x_0, promedia los valores de $T_i(x_0)$

Random Forest

Debido a que son aleatorios, cada árbol es un predictor débil de manera individual, aunque decente, pues los splits son escogidos de manera óptima.

Gracias a la ley de los grandes números y al teorema central de límite, el ensamble es un predictor mucho más fuerte que cualquiera individual, con pocas posibilidades de causar sobreajuste.

Este modelo no impone ninguna restricción sobre el conjunto de datos, por lo que es una elección ideal como primer modelo. Muchas veces es muy difícil encontrar un mejor predictor a pesar de pasar una gran cantidad de tiempo realizando transformaciones y probando diferentes modelos.

State of the Art

Existen métodos más sofisticados, que imponen una serie de supuestos en el conjunto de datos, o requieren una serie de transformaciones para ser efectivos. Support Vector Machines, Boosting, Modelos Gráficos, entre otros.

Estos modelos son muy capaces de hacer un mejor trabajo en casos específicos, pero es necesario entender los supuestos y ser capaz de verificarlos para nuestro conjunto de datos. Muchas veces eso va más allá de simples mediciones numéricas.

Por último, en una escala muy grande, todos los modelos tienen *performance* similar, gracias a la ley de los grandes números. Cada problema es único y requiere una exploración del conjunto de datos y entendimiento del problema a resolver.