Programación Estadística: Más funciones

Jocelyn Simmonds (jsimmond@dcc.uchile.cl)

Departamento de Ciencias de la Computación

Modelos lineales

Cargando unos datos

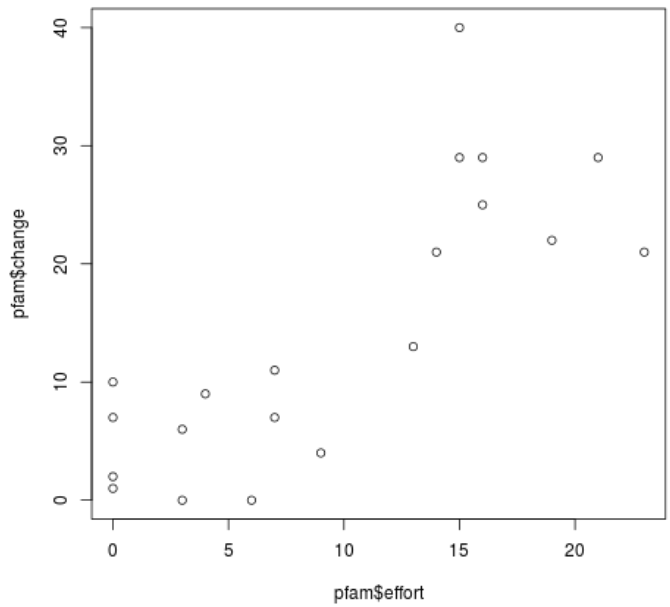
Primero cargaremos unos datos acerca de iniciativas de planificación familiar de países latinoamericanos:

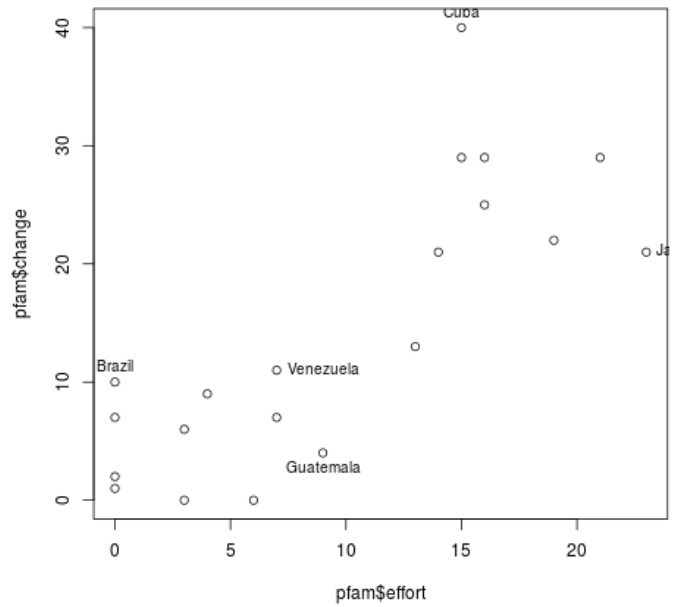
```
> pfam <- read.table("effort.dat")
> head(pfam, 3)
        setting effort change
Bolivia      46      0      1
Brazil       74      0     10
Chile        89     16     29
> summary(pfam)
    setting         effort          change
 Min.   :35.0   Min.   : 0.00   Min.   : 0.00
 1st Qu.:66.0   1st Qu.: 3.00   1st Qu.: 5.50
 Median :74.0   Median : 8.00   Median :10.50
 Mean   :72.1   Mean   : 9.55   Mean   :14.30
 3rd Qu.:84.0   3rd Qu.:15.25   3rd Qu.:22.75
 Max.   :91.0   Max.   :23.00   Max.   :40.00
>
```

Examinando los datos

Podemos usar la función `identify()` para identificar puntos de los gráficos en forma interactiva:

```
1  > plot(pfam$effort, pfam$change)
2  > identify(pfam$effort, pfam$change, row.names(pfam), ps=10)
3  #          elegir algunos puntos del grafico (este debe estar activo)
4  #          hacer right-click para terminar de identificar puntos
5  [1]  2  6 10 13 20
```





Modelo lineal

La función `lm()` genera un modelo lineal de los datos:

```
1   > modelo = lm(pfam$change ~ pfam$setting + pfam$effort)
```

El parámetro principal de `lm()` es una formula: la variable a la izquierda del ~ se modela en base a la formula a la derecha. Pueden usar los siguientes operadores:

- $+$ para combinar variables: $A + B$
- : para incluir interacciones entre variables: $A : B$
- * para incluir variables y sus interacciones: $A * B$, que es equivalente a $A + B + A : B$

A y B tambien pueden ser factores.

Resumen de un modelo lineal

Pueden ver un resumen del modelo generado:

```
> summary(modelo)
Call:
lm(formula = pfam$change ~ pfam$setting + pfam$effort)

Residuals:
     Min       1Q   Median       3Q      Max
-10.3475  -3.6426   0.6384   3.2250  15.8530

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -14.4511     7.0938  -2.037 0.057516 .
pfam$setting   0.2706     0.1079   2.507 0.022629 *
pfam$effort    0.9677     0.2250   4.301 0.000484 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.389 on 17 degrees of freedom
Multiple R-squared:  0.7381, Adjusted R-squared:  0.7073
F-statistic: 23.96 on 2 and 17 DF,  p-value: 1.132e-05
```

Elementos de un modelo lineal

Pueden hacer más que graficar el modelo lineal:

```
> esfuerzo <- fitted(modelo)
> esfuerzo
        1         2         3         4         5         6         7         8
-2.004026  5.572452 25.114699 21.867637 28.600325 24.146986 17.496913 10.296380
        9        10        11        12        13        14        15        16
14.364491  9.140694 -2.077359  6.122912 31.347518 11.878604  3.948921 26.664898
       17        18        19        20
 8.475593  5.301864 22.794043 16.946453
> coef(modelo)
 (Intercept) pfam$setting  pfam$effort
 -14.4510978    0.2705885    0.9677137
```

Usando factores

También pueden crear modelos lineales usando factores:

```
> effort_fact <- cut(pfam$effort, breaks = c(-1, 4, 14, 100),
+ label=c("weak","moderate","strong"))
> effort_fact
 [1] weak     weak     strong   strong   strong   strong   moderate moderate
 [9] moderate moderate weak     moderate strong   weak     weak     strong
[17] weak     weak     strong   moderate
Levels: weak moderate strong
> nuevo_modelo <- lm(pfam$change ~ pfam$setting + effort_fact)
> nuevo_modelo
Call:
lm(formula = pfam$change ~ pfam$setting + effort_fact)

Coefficients:
        (Intercept)         pfam$setting  effort_factmoderate
            -5.9540               0.1693               4.1439
  effort_factstrong
            19.4476
```

El primer nivel del factor es el valor de referencia.

Usando factores

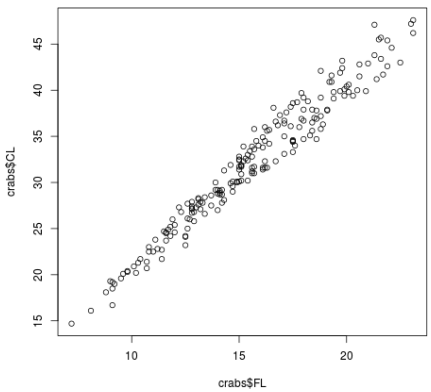
```
> summary(nuevo_modelo)

Call:
lm(formula = pfam$change ~ pfam$setting + effort_fact)

Residuals:
     Min       1Q   Median       3Q      Max
-10.0386  -2.8198   0.1036   1.3269  11.4416

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)          -5.9540     7.1660  -0.831    0.418
pfam$setting          0.1693     0.1056   1.604    0.128
effort_factmoderate   4.1439     3.1912   1.299    0.213
effort_factstrong    19.4476     3.7293   5.215 8.51e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.732 on 16 degrees of freedom
Multiple R-squared:  0.8016, Adjusted R-squared:  0.7644
F-statistic: 21.55 on 3 and 16 DF,  p-value: 7.262e-06
```

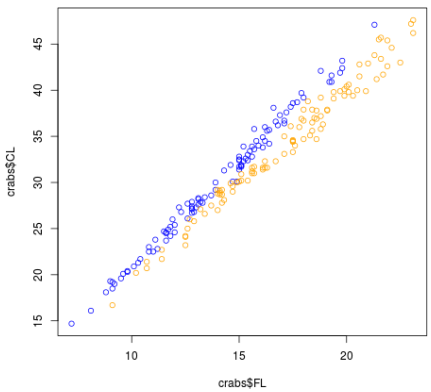
Más plots

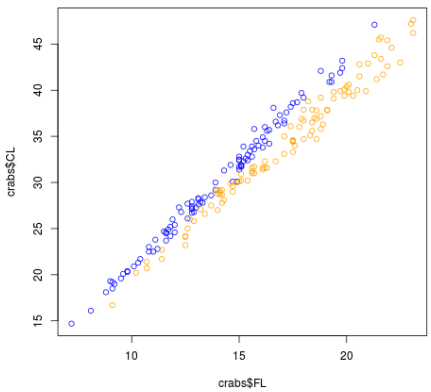
Gráficos con colores

Definir colores en base a algún factor:

```
> library(MASS)
> head(crabs, n = 1)
  sp sex index  FL  RW   CL   CW  BD
1  B   M     1 8.1 6.7 16.1 19.0 7.0
> plot(crabs$FL, crabs$CL)                    # primer grafico
> colores <- c("blue", "orange")[crabs$sp]
> head(colores, n = 3)
[1] "blue" "blue" "blue"
> tail(colores, n = 3)
[1] "orange" "orange" "orange"
> plot(crabs$FL, crabs$CL, col=colores)       # tercer grafico
> plot(crabs$CL ~ crabs$FL, col=colores)      # segundo grafico
```



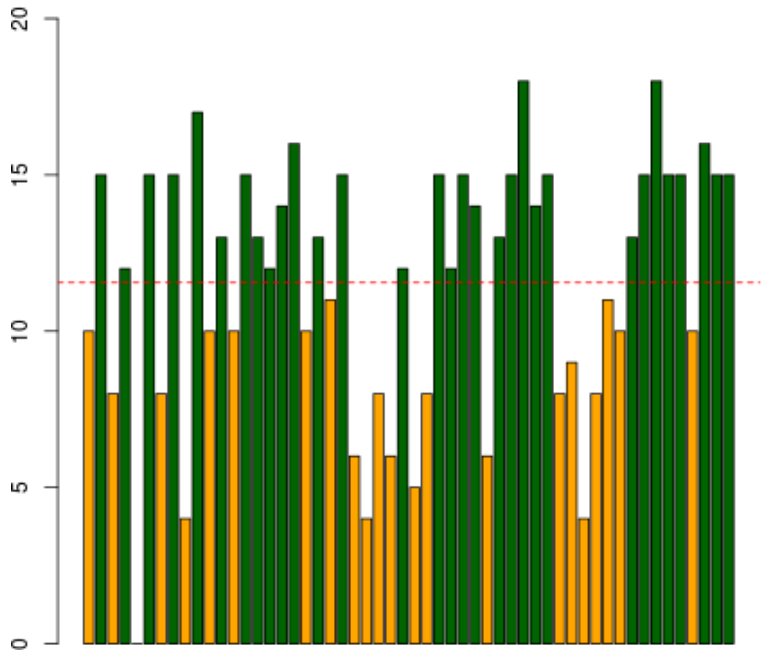




Gráficos con colores

Definir colores en base a alguna formula sobre los datos:

```
> malos_comp <- ifelse(painters$Composition < mean(painters$Composition),
                       "orange", "darkgreen")
> malos_comp
 [1] "orange"    "darkgreen" "orange"    "darkgreen" "orange"    "darkgreen"
...
> barplot(painters$Composition, col = malos_comp, ylim = c(0, 20))
> abline(h = mean(painters$Composition), lty = 2, col = "red")
```



Lineas de tendencia

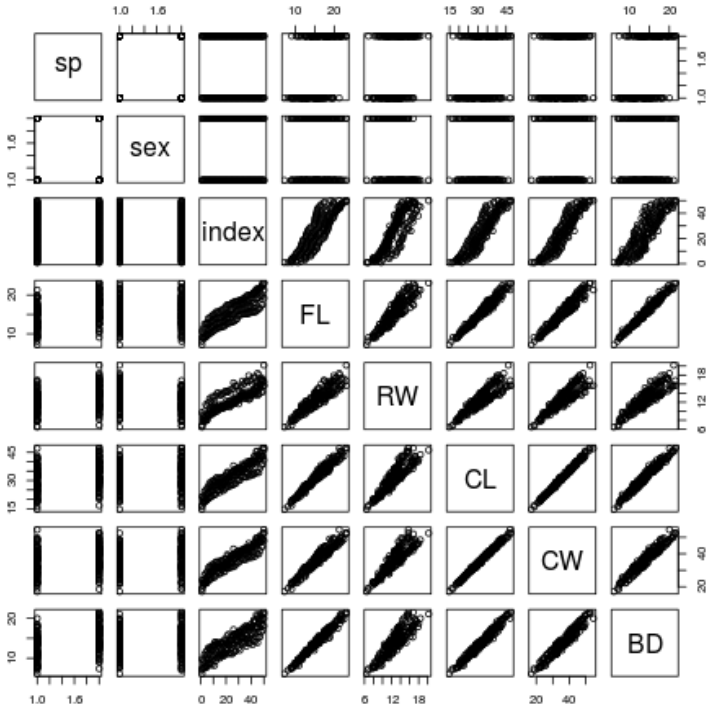
La función `abline` recibe varios parametros:

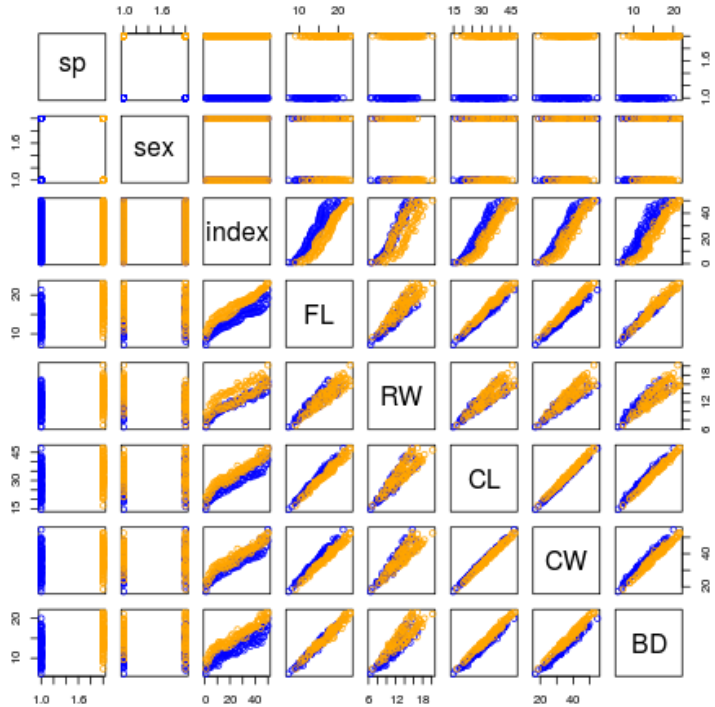
- `h = n`: linea horizontal que corta al eje Y en el valor n
- `v = n`: linea vertical que corta al eje X en el valor n
- `a = n, b = m`: linea $y = a + bx$
- `lty = n`: blank (0), solid (1), dashed (2), dotted (3), dotdash (4), longdash (5), twodash (6). Valor por defecto: 1
- `lwd = n`: $n > 0$ es el ancho de la linea
- `col = color`: red, green, etc.

Gráficos resumen

Pueden usar `plot()` o `pairs()` para generar gráficos de resumen:

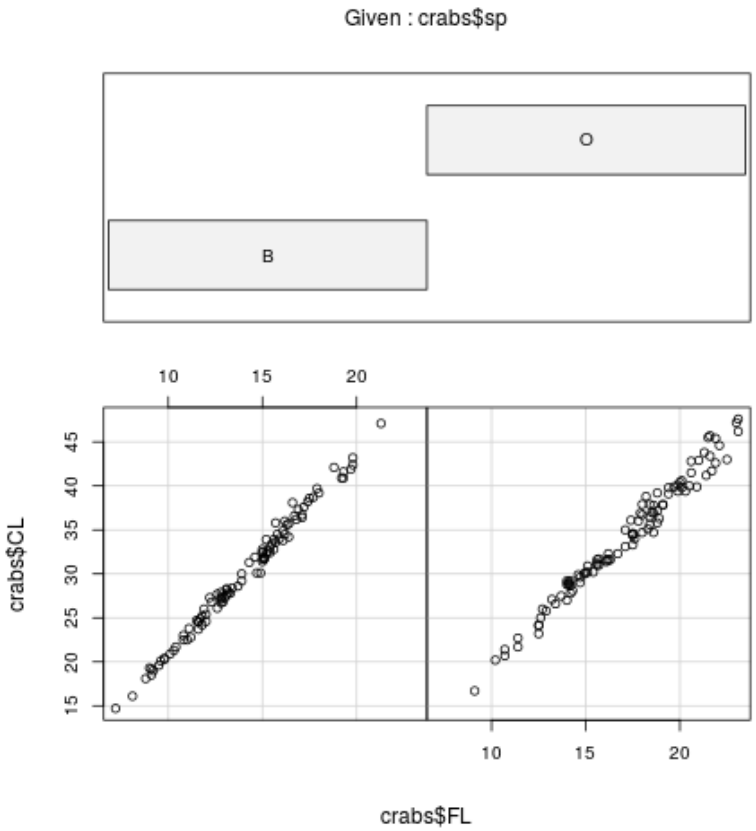
```
> colores <- c("blue", "orange")[crabs$sp]
> plot(crabs)                          # primer grafico
> plot(crabs, col=colores)             # segundo grafico
```

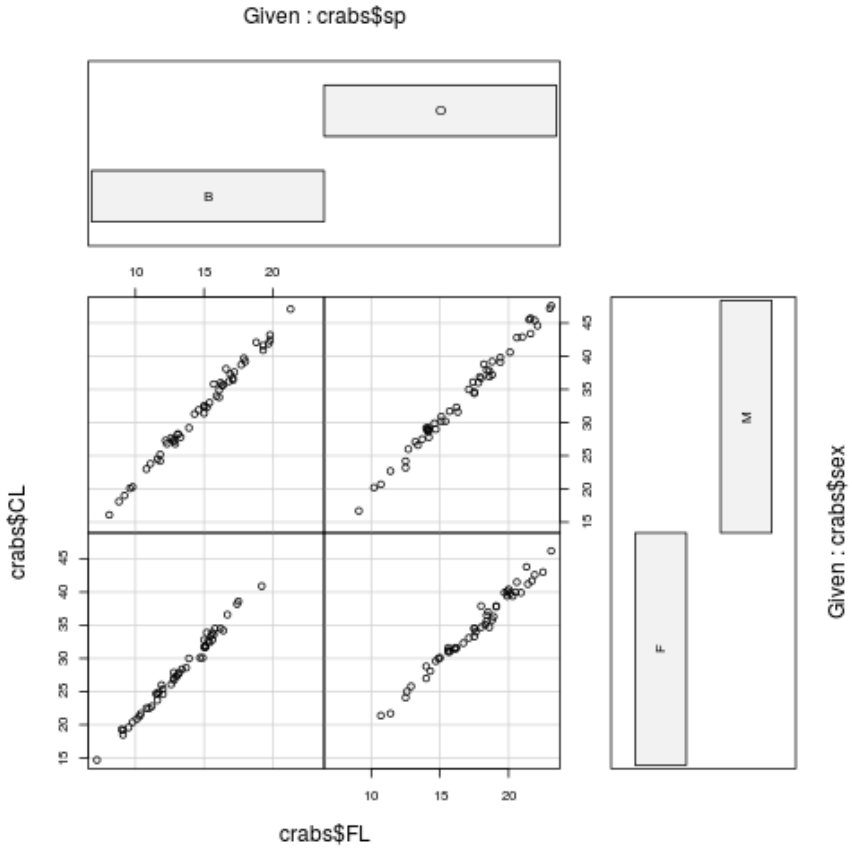




Gráficos resumen

```
1  > coplot(crabs$CL ~ crabs$FL | crabs$sp)                  # primer grafico
2  > coplot(crabs$CL ~ crabs$FL | crabs$sp + crabs$sex)      # segundo grafico
```





Strings

Expresiones regulares

Si un vector contiene strings, pueden buscar valores usando expresiones regulares:

- "`.`" coincide con cualquier carácter (*comodín*)
- "`?`" encontrar si algo aparece 0 ó 1 veces
- "`*`" encontrar si algo aparece 0 ó más veces
- "`+`" encontrar si algo aparece 1 ó más veces
- "`{n}`" encontrar si algo aparece exactamente *n* veces
- "`{n,}`" encontrar si algo aparece *n* o más veces
- "`{n,m}`" encontrar si algo aparece al menos *n* veces, pero a lo más *m* veces

Expresiones regulares

Pueden usar la función `grep()` para encontrar los elementos de un vector que coinciden con la expresión regular:

```
> x <- c("abc", "def", "cba a", "aa")
> grep("a+", x)
[1] 1 3 4
> grep("a+", x, value = TRUE)
[1] "abc"   "cba a" "aa"
> grep("a?", x, value = TRUE)
[1] "abc"   "def"   "cba a" "aa"
> grep("aa", x, value = TRUE)
[1] "aa"
> grep("aa*", x, value = TRUE)
[1] "abc"   "cba a" "aa"
> grep("aa+", x, value = TRUE)
[1] "aa"
```

Ojo: `ignore.case = FALSE` por defecto

Expresiones regulares

Pueden usar la función `grep()` para encontrar los elementos de un vector que coinciden con la expresión regular:

```
> adn <- c("ATCGCGAATTCAC", "ATGACGTACGTACGACTG",
           "ACTGCATTATATCGTACGAAATTATACGCGCG")
> grep("GCGA", adn, value = TRUE)
> grep("G.G", adn, value = TRUE)
> grep("G(.){2,3}G", adn, value = TRUE)
> grep("G(.){6,8}G", adn, value = TRUE)
[1] "ATGACGTACGTACGACTG"
> grep("(TA){1,2}", adn, value = TRUE)
[1] "ATGACGTACGTACGACTG"                "ACTGCATTATATCGTACGAAATTATACGCGCG"
> grep("(TA){2}", adn, value = TRUE)
[1] "ACTGCATTATATCGTACGAAATTATACGCGCG"
```

Expresiones regulares: clases de caracteres

R define algunas clases de caracteres para definir expresiones regulares:

- "`[:lower:]`" letras minúsculas en el locale actual
- "`[:upper:]`" letras mayúsculas en el locale actual
- "`[:alpha:]`" = `[:lower]` + `[:upper]`
- "`[:digit:]`" números 0, 1, ...9
- "`[:alnum:]`" = `[:alpha]` + `[:digit]`
- "`[:punct:]`" signos de puntuación
- "`[:space:]`" espacio, tab, linea nueva, etc.
- "`[ABC]`" también pueden definir sus propias clases de caracteres

Expresiones regulares: clases de caracteres

Busquemos las IP en el rango 192.168.1.0 - 192.168.1.255:

```
> ip <- c("192.168.1.45", "205.34.4.34", "200.89.69.134",
          "192.168.1.45", "123.168.1.1")
> grep("192\\.168\\.1\\.[[:digit:]]{1,3}", ip, value = TRUE)
[1] "192.168.1.45" "192.168.1.45"
```

Ahora buscamos todas las IP de la forma 2XX.XXX.XXX.XXX:

```
> grep("2[[:digit:]]{2}\\.[[:digit:]]{1,3}\\.
      [[:digit:]]{1,3}\\.[[:digit:]]{1,3}", ip, value = TRUE)
[1] "205.34.4.34"   "200.89.69.134"
```

Tenemos que usar el doble \ para buscar coincidencias con el carácter “.”, deben hacer lo mismo para encontrar coincidencias con los otros caracteres especiales.

Expresiones regulares: caracteres de control

R define algunos caracteres de control:

- "^" busca coincidencias al inicio del string
- "$" busca coincidencias al final del string
- "[^ab]" cualquier caracter menos a y b
- "a|b" busca coincidencia de a o b

Ejemplos:

```
1  > grep("(192|123)\\.168\\.1\\.[[:digit:]]{1,3}", ip, value = TRUE)
2  [1] "192.168.1.45" "192.168.1.45" "123.168.1.1"
3  > nombres <- c("Isabella", "Daniel", "Olivia", "David", "Alexis",
4                 "Gabriel", "Sofia", "Benjamin")
5  > grep("a$", nombres, value = TRUE)
6  [1] "Isabella" "Olivia"   "Sofia"
```

Usando expresiones regulares

Otras funciones que toman como parámetro una expresión regular:

- `regexpr(exp_reg, vector)`: retorna la posicion de la primera ocurrencia de `exp_reg` en cada elemento de `vector`
- `gregexpr(exp_reg, vector)`: retorna las posiciones de todas las ocurrencias de `exp_reg` en cada elemento de `vector`
- `sub(exp_reg, sustituto, vector)`: sustituye la primera sub-cadena del string que coincide con la `exp_reg` usando el valor `sustituto` (para cada elemento de `vector`).
- `strsplit(string, exp_reg)`: corta el string en cada coincidencia de `exp_reg`

Ejemplos

```
> adn
[1] "ATCGCGAATTCAC"
[2] "ATGACGTACGTACGACTG"
[3] "ACTGCATTATATCGTACGAAATTATACGCGCG"
>  grep("G(.){2,3}G", adn, value = TRUE)
[1] "ATGACGTACGTACGACTG"
[2] "ACTGCATTATATCGTACGAAATTATACGCGCG"
> regexpr("G(.){2,3}G", adn)
[1] -1  3 14
attr(,"match.length")
[1] -1  4  5
attr(,"useBytes")
[1] TRUE
```

```
> gregexpr("G(.){2,3}G", adn)
[[1]]
[1] -1
attr(,"match.length")
[1] -1
attr(,"useBytes")
[1] TRUE

[[2]]
[1]  3 10
attr(,"match.length")
[1] 4 5
attr(,"useBytes")
[1] TRUE

[[3]]
[1] 14 28
attr(,"match.length")
[1] 5 5
attr(,"useBytes")
[1] TRUE
```

Ejemplos

Usando `sub()`:

```
> nombres
[1] "Isabella" "Daniel"   "Olivia"   "David"    "Alexis"   "Gabriel"  "Sofia"
[8] "Benjamin"
> sub("a$", "e", nombres)
[1] "Isabelle" "Daniel"   "Olivie"   "David"    "Alexis"   "Gabriel"  "Sofie"
[8] "Benjamin"
```

Usando `strsplit()`:

```
> adn
[1] "ATCGCGAATTCAC"                    "ATGACGTACGTACGACTG"
[3] "ACTGCATTATATCGTACGAAATTATACGCGCG"
> strsplit(adn[3], "TA")
[[1]]
[1] "ACTGCAT" ""        "TCG"     "CGAAAT"  ""        "CGCGCG"
```

Más funciones sobre strings

- `nchar(string)`: largo del `string`
- `substr(string, pos_inicial, pos_final)`: retorna parte del `string`, desde `pos_inicial` hasta (incluyendo) `pos_final` (o largo de `string`, cualquiera sea menor).
- `tolower(string)`: retorna `string` en minusculas
- `toupper(string)`: retorna `string` en mayusculas

Ejemplos:

```
1  > nchar(adn[3])
2  [1] 32
3  > substr(adn[3], 3, 6)
4  [1] "TGCA"
5  > tolower("ABCdef")
6  [1] "abcdef"
7  > toupper("ABCdef")
8  [1] "ABCDEF"
```