caracteres y datos de tiempo

Santiago Lozano

7 de marzo de 2020

Texto y caracteres

Recordemos que en R los datos de tipo `character` son definidos en comillias

```
a <- "abc"
b <- "123"
```

de igual forma sabemos que los datos de tipo `numeric` pueden ser transformados a `character`, pero los `character` de tipo no numérico no pueden ser tranformados a `numeric`

```
as.numeric(a)
```

```
NAs introduced by coercion[1] NA
```

```
as.numeric(b)
```

```
## [1] 123
```

Debemos tener una distinción entre `length()` que especifica el largo de un vector y `nchar()` el cual despliega el número de caracteres que tiene un `character`

```
mascotas<- c("gato","perro","conejo","hamster")
length(mascotas)
```

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

```
nchar(mascotas)
```

```
## [1] 4 5 6 7
```

Y también debemos hacer una distinción entre vectores de tipo `character` y `factor`, los últimos, los cuales establence una variable de tipo cualitativo

```
class(mascotas)
```

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

```
is.factor(mascotas)
```

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

R tiene preestablecido vectores que tienen las letras del abecedario en minúscula

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

Y mayúscula

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

Podemos desplegar los datos `character` si comillas

```
noquote(letters)
```

```
##  [1] a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Y especificar cuál es la posición de cierto elemento explícito de un vector

```
which(letters=="n")
```

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

Concatenación de datos de tipo `character` usamos distintas funciones

```
c(a,b)
```

```
## [1] "abc" "123"
```

la cual despliega en un vector que concatena los dos vectores o datos, otra forma es con

```
paste(a,b,sep = "")
```

```
## [1] "abc123"
```

```
paste(a,b)
```

```
## [1] "abc 123"
```

Es importante mencionar que cada dato `character` en una estructura de datos no pierde su espacio en blanco cuando se le aplica `paste()`, y se especifica `sep=""`

```
paste(a,b," una frase más larga conteniedo espacios en blanco",sep="")
```

```
## [1] "abc123 una frase más larga conteniedo espacios en blanco"
```

Si uno de los argumentos de `paste()` es un vector, cada uno de los elementos del vector es es pegado junto con el resto de los argumentos

```
d <- c(a,b,"nuevo")
e <- paste(d,"una frase más larga que contiene espacios en blanco")
e
```

```
## [1] "abc una frase más larga que contiene espacios en blanco"
## [2] "123 una frase más larga que contiene espacios en blanco"
## [3] "nuevo una frase más larga que contiene espacios en blanco"
```

Si por ejemplo, queremos especificar varios elementos que están en una carpeta

```
drive <- "c:"
carpeta <- "programcion"
archivo <- c("archivo1","archivo2","archivo3")
extension <- ".txt"
paste(drive, "\\",carpeta, "\\",archivo, extension,sep="")
```

```
## [1] "c:\\programcion\\archivo1.txt" "c:\\programcion\\archivo2.txt"
## [3] "c:\\programcion\\archivo3.txt"
```

caso de dos vectores

```
string1 <- c("a","ab","B","bba","one","!@","brute")
string2 <- c("e","D")
paste(string1,string2)
```

```
## [1] "a e"     "ab D"    "B e"     "bba D"   "one e"   "!@ D"    "brute e"
```

```
paste(string1,string2,sep="-")
```

```
## [1] "a-e"     "ab-D"    "B-e"     "bba-D"   "one-e"   "!@-D"    "brute-e"
```

```
paste(string1,string2,collapse="",sep="")
```

```
## [1] "aeabDBebbaDonee!@Dbrutee"
```

donde `collapse=` es usado para concatenar elementos dentro del dato `character`

Extracción de partes de un character

```
frase <- "el rápido gato marrón salta sobre el perro perezoso"
```

la función `substr()` se usa para extraer letras de un número especificado de caracteres con un output de `character`

```
substr(frase,1,1)
```

```
## [1] "e"
```

Aquí se extrajo una letra desde la primera letra

```
substr(frase,2,4)
```

```
## [1] "l r"
```

```
q <- character(20)
for(i in 1:20){
  q[i] <- substr(frase,1,i)
}
q
```

```
##  [1] "e"                   "el"                  "el "
##  [4] "el r"                "el rá"               "el ráp"
##  [7] "el rápi"             "el rápid"            "el rápido"
## [10] "el rápido "          "el rápido g"         "el rápido ga"
## [13] "el rápido gat"       "el rápido gato"      "el rápido gato "
## [16] "el rápido gato m"    "el rápido gato ma"   "el rápido gato mar"
## [19] "el rápido gato marr" "el rápido gato marró"
```

Separación de Characters

Para separar elementos de un `character` (incluyendo espacios en blano) de tipo de dato, usamos `strsplit()`

```
strsplit(frase,split = "")
```

```
## [[1]]
##  [1] "e" "l" " " " " "r" "á" "p" "i" "d" "o" " " " " "g" "a" "t" "o" " " " " "m" "a" "r" "r"
## [20] "ó" "n" " " " " "s" "a" "l" "t" "a" " " " " "s" "o" "b" "r" "e" " " " " "e" "l" " " " " "p"
## [39] "e" "r" "r" "o" " " " " "p" "e" "r" "e" "z" "o" "s" "o"
```

la función `table` puede ser usada para contar las ocurrencias de cada letra

```
table(strsplit(frase,split = ""))
```

```
##
##   a á b d e g i l m n o ó p r s t z
## 8 4 1 1 1 6 1 1 3 1 1 6 1 3 7 3 2 1
```

Para separar por palabras

```
strsplit(frase, " ")
```

```
## [[1]]
## [1] "el"       "rápido"   "gato"     "marrón"   "salta"    "sobre"    "el"
## [8] "perro"    "perezoso"
```

contando el número de letras de cada palabra

```
lapply(strsplit(frase, " "), nchar)
```

```
## [[1]]
## [1] 2 6 4 6 5 5 2 5 8
```

Revertiendo el orden de las letras

```
lapply(strsplit(frase,""),rev)
```

```
## [[1]]
##  [1] "o" "s" "o" "z" "e" "r" "e" "p" " " "o" "r" "r" "e" "p" " " "l" "e" " " "e"
## [20] "r" "b" "o" "s" " " "a" "t" "l" "a" "s" " " "n" "ó" "r" "r" "a" "m" " " "o"
## [39] "t" "a" "g" " " "o" "d" "i" "p" "á" "r" " " "l" "e"
```

```
sapply(lapply(strsplit(frase, NULL), rev), paste, collapse="")
```

```
## [1] "osozerep orrep le erbos atlas nórram otag odipár le"
```

Para usar una palabra como marca de separación

```
strsplit(frase,"el")
```

```
## [[1]]
## [1] ""                                " rápido gato marrón salta sobre "
## [3] " perro perezoso"
```

Y extrayendo un elemento de esta lista

```
strsplit(frase,"el")[[1]][2]
```

```
## [1] " rápido gato marrón salta sobre "
```

Transformación de textos en mayúscula y minúscula

```
toupper(frase)
```

```
## [1] "EL RÁPIDO GATO MARRÓN SALTA SOBRE EL PERRO PEREZOSO"
```

```
tolower(frase)
```

```
## [1] "el rápido gato marrón salta sobre el perro perezoso"
```

Función match()

Esta función responde la tregunta ¿Dónde los valores en el segundo vector aparecen en el primer vector

```
primero <- c(5,8,3,5,3,6,4,4,2,8,8,8,4,4,6)
segundo <- c(8,6,4,2)
match(primero,segundo)
```

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

vemos que aquí la forma en cómo trabaja la función, así, ¡Dónde el 5 del primer vector aparece en el segundo vector?, no aparece, entonces R despiega NA en el output,¿dónde el 8 del primer vector aparece en el segundo? aparece en el primero, y el programa despliega 1.

Esta función es muy importante a la hora de clasificación en base de datos

Por ejemplo, en el campo de la mediciona usted tiene un vector que contiene las identificaciones anónimas de nueve pacientes

```
sujetos <- c("A", "B", "G", "M", "N", "S", "T", "V", "Z")
```

Suponga que usted quiere probar una nueva droga a todos los pacientes en un vector de pacientes seleccionados

```
pacientes.selec<- c("E", "G", "S", "U", "Z")
```

Usted quiere buscar estos pacientes en la base de datos general para obtener algunos datos, así podemos usar `match()` para buscarlos en la base de datos

```
match(sujetos,pacientes.selec)
```

```
## [1] NA NA  2 NA NA  3 NA NA  5
```

Queremos obtener esta información sabiendo qué paciente se le da la droga nueva y cuál la convencional

```
q<- character(length(match(sujetos,pacientes.selec)))
droga <- c("nueva" , "convencional")
for(i in 1:length(match(sujetos,pacientes.selec))){
  if(is.na(match(sujetos,pacientes.selec))[i]){
    q[i]<-droga[2]
  }else{
    q[i]<-droga[1]
  }
}
q
```

```
## [1] "convencional" "convencional" "nueva"        "convencional" "convencional"
## [6] "nueva"        "convencional" "convencional" "nueva"
```

Identificación de patrones

importemos el siguiente dataframe

```
setwd("C:/Users/santiago/Documents/Progrmación en R/2020-I/PR06-caracteres y datos de tiempo")
worldfloras<-read.table("worldfloras.txt",header=T)
attach(worldfloras)
names(worldfloras)
```

```
## [1] "Country"    "Latitude"   "Area"       "Population" "Flora"
## [6] "Endemism"   "Continent"
```

Existe una forma de encontrar patrones en un vector y apartir de este poder extraer información del dataframe, esto se realiza mediante la función `grep`

en el dataframe tenemos una variable `Country` que corresponde a ciertos países, el cual es un tió factor pues está en un dataframe

```
head(Country)
```

```
## [1] Afghanistan Albania     Algeria     Andorra     Angola      Antarctica
## 161 Levels: Afghanistan Albania Algeria Andorra Angola Antarctica ... Zimbabwe
```

Para extraer la ubicación de los países que Tienen la letra R

```
grep("R",as.character(Country))
```

```
## [1]  27  34  40 116 118 119 120 152
```

Con lo que, extrayendo los nombres de los países

```
as.vector(Country[grep("R",as.character(Country))])
```

```
## [1] "Central African Republic" "Costa Rica"
## [3] "Dominican Republic"       "Puerto Rico"
## [5] "Reunion"                  "Romania"
## [7] "Rwanda"                   "USSR"
```

Si queremos los países que comienzan con R

```
as.vector(Country[grep("^R",as.character(Country))])
```

```
## [1] "Reunion" "Romania" "Rwanda"
```

Para seleccionar los países cuyos nombres son más de una palabra y la que nos esprimera palabra comienza con R

```
as.vector(Country[grep(" R",as.character(Country))])
```

```
## [1] "Central African Republic" "Costa Rica"
## [3] "Dominican Republic"       "Puerto Rico"
```

Para encontrar los países con dos o más palabras

```
head(as.vector(Country[grep(" ",as.character(Country))]))
```

```
## [1] "Balearic Islands"         "Burkina Faso"
## [3] "Central African Republic" "Costa Rica"
## [5] "Dominican Republic"       "El Salvador"
```

Los países cuyo nombre finaliza en y, usamos $

```
as.vector(Country[grep("y$",as.character(Country))])
```

```
## [1] "Hungary"  "Italy"    "Norway"   "Paraguay" "Sicily"   "Turkey"   "Uruguay"
```

Para condiciones que impliquen expresiones en grupo, es decir, un serie de números o letras ubicadas alfabéticamente, usamos "[]" dentro de los cuales indicamos el rango de valores o letras que son seleccionados. Entonces, para seleccionar los países cuyo nombre contiene las letras de C a E mayúsculas hacemos

```
head(as.vector(Country[grep("[C-E]",as.character(Country))]))
```

```
## [1] "Cameroon"                 "Canada"
## [3] "Central African Republic" "Chad"
## [5] "Chile"                    "China"
```

Aquí hay países como Ivory Coast o Trinstan Cunha que contienen la letra pero no es la primera, para estos propósitos usamos

```
head(as.vector(Country[grep("^[C-E]",as.character(Country))]))
```

```
## [1] "Cameroon"                 "Canada"
## [3] "Central African Republic" "Chad"
## [5] "Chile"                    "China"
```

Para mencionar los países que no cumplen cierta condición usamo el menos "-" , así, buscamos los países cuta palabra no finaliza en una letra de la a la t

```
as.vector(Country[-grep("[a-t]$",as.character(Country))])
```

```
##  [1] "Hungary"  "Italy"    "Norway"   "Paraguay" "Peru"     "Sicily"
##  [7] "Turkey"   "Uruguay"  "USA"      "USSR"     "Vanuatu"
```

Si queremos del output anterior quitar USA y USSR podemos poner varias condiciones

```
as.vector(Country[-grep("[A-T a-t]$",as.character(Country))])
```

```
## [1] "Hungary"  "Italy"    "Norway"   "Paraguay" "Peru"     "Sicily"   "Turkey"
## [8] "Uruguay"  "Vanuatu"
```

El punto en R es un símbolo que es considerado para condicionar cualquier caracter, así, por ejemplo, si queremos especificar la palabra cuya letra es "y" hacemos ^.y, es decir comenzando con cualquier palabra que la siguiente sea "y"

```
as.vector(Country[grep("^.y",as.character(Country))])
```

```
## [1] "Cyprus" "Syria"
```

para los países con y como segunda letra ^..y

```
as.vector(Country[grep("^..y",as.character(Country))])
```

```
## [1] "Egypt"      "Guyana"     "Seychelles"
```

Si queremos que y sea la sexta letra

```
as.vector(Country[grep("^.{5}y",as.character(Country))])
```

```
## [1] "Norway" "Sicily" "Turkey"
```

entonces el .{5} menciona 5 caracteres cualesquiera,

```
as.vector(Country[grep("^.{,4}y",as.character(Country))])
```

```
##  [1] "Cyprus"      "Egypt"       "Guyana"      "Italy"       "Ivory Coast"
##  [6] "Kenya"       "Libya"       "Malaysia"    "Norway"      "Seychelles"
## [11] "Sicily"      "Syria"       "Turkey"
```

vemos que "." significa cualquier caracter, mientras que $\{,4\}$ significa repetir hasta 4 cualquier cosa antes del $ (el string final). Así, todos los países con 15 o más caracteres los filtramos mediante

```
as.vector(Country[grep("^.{15,}$",as.character(Country))])
```

```
## [1] "Balearic Islands"         "Central African Republic"
## [3] "Dominican Republic"       "Papua New Guinea"
## [5] "Solomon Islands"          "Trinidad & Tobago"
## [7] "Tristan da Cunha"
```

Subtitucion de text en un character

R tiene la posibilidad de buscar un caracter y realizar una operación que reemplace ese caracter mediante las funciones `sub()` y `gsub()`, funciones las cuales difieren sólo en que `sub()` reemplaza sólo la primera ocurrencia y `gsub()` reemplaza todas las ocurrencias, veamos un ejemplo

```
text <- c("arm", "leg", "head", "foot", "hand", "hindleg", "elbow")
```

y queremos reemplazar "h" por "H"

```
gsub("h","H",text)
```

```
## [1] "arm"     "leg"     "Head"    "foot"    "Hand"    "Hindleg" "elbow"
```

Si queremos sólo reemplazar la primera ocurrencia

```
sub("o","O",text)
```

```
## [1] "arm"     "leg"     "head"    "fOot"    "hand"    "hindleg" "elbOw"
```

```
gsub("^.","O",text)
```

```
## [1] "Orm"     "Oeg"     "Oead"    "Ooot"    "Oand"    "Oindleg" "Olbow"
```

Para este dividimos en 2 grupos, en primer grupo al mencionar `\\w` los caracteres que son letras despues `?` completantdo con frecuencia una vez, es decir, `(\\w?)` me agrupa un grupo de la primera letra y el segundo al poner `\*` agregar al menos un item, es decir, `\\w*` la cadena de al menos cero letras

```
gsub("(\\w?)(\\w*)", "\\U\\1\\L\\2",text, perl=TRUE)
```

```
## [1] "Arm"     "Leg"     "Head"    "Foot"    "Hand"    "Hindleg" "Elbow"
```

```
gsub("(\\w*)", "\\U\\1",text, perl=TRUE)
```

```
## [1] "ARM"     "LEG"     "HEAD"    "FOOT"    "HAND"    "HINDLEG" "ELBOW"
```

Localización de un patrón en un vector usando regexpr

En vez de sustituir el patrón, podemos querer saber si este ocurren en un string, y si es así, donde ocurre dentro de cada string. El resultado de `regexpr` es por tanto un vector numeric, pero indicando la posición de la primera instancia del patrón en el string. Si el patrón no aparece en el string, el valor por defecto retornado por `regexpr` es `-1`

```
text
```

```
## [1] "arm"     "leg"     "head"    "foot"    "hand"    "hindleg" "elbow"
```

```
regexpr("o",text)
```

```
## [1] -1 -1 -1  2 -1 -1  4
## attr(,"match.length")
## [1] -1 -1 -1  1 -1 -1  1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

Podemos indicar que despliegue cero si el string no aparece en y un número dependiento la posición de la primera ocurrencia del string

```r
gregexpr("o",text)
```

```
## [[1]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[3]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[4]]
## [1] 2 3
## attr(,"match.length")
## [1] 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[5]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[6]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
```

```
## [1] TRUE
##
## [[7]]
## [1] 4
## attr(,"match.length")
## [1] 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

Desplegando una lista donde cada elemento de la lista es un vector el cual, el primer elemento despliega las posiciones de las ocurrencias del patrón, el segundo el largo del texto encontrado, el tercero el tipo de dato y el cuarto su usa bytes

```
lapply(gregexpr("o",text),length)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 1
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 1
##
## [[6]]
## [1] 1
##
## [[7]]
## [1] 1
```

el cual genera el número de ocurrencias del patrón

```
freq <- as.vector(unlist(lapply(gregexpr("o",text),length)))
present <- ifelse(regexpr("o",text)<0,0,1)
freq*present
```

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

Indicando cero si no hay match y un número distinto de cero indicando la cantidad de ocurrencias del patrón

La función `charmatch` es para realizar match entre caracteres, buscando si el patrón se encuentran en alguna parte del elemento del vector, y despliega cero si hay múltiple matches

```
charmatch("m", c("mean", "median", "mode"))
```

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

```
charmatch("med", c("mean", "median", "mode"))
```

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

Uso de %in% y which

Si queremos ver matches entre vectores de caracteres

```
stock <- c("carro","van")
requisitos <- c("camión","remolque","van","deportivo","carro","vagón","carro")
```

Usamos `which` si queremos encontrar la ubicación del primer vector nombrado en cualesquieras de las entradas del segundo vector nombrado

```
which(requisitos %in% stock)
```

```
## [1] 3 5 7
```

Si queremos saber que elementos son

```
requisitos[which(requisitos %in% stock)]
```

```
## [1] "van"   "carro" "carro"
```

y para saber la ubicación exacta

```
sapply(requisitos, "%in%", stock)
```

```
##    camión  remolque       van deportivo     carro     vagón     carro
##     FALSE     FALSE      TRUE     FALSE      TRUE     FALSE      TRUE
```

```
requisitos %in% stock
```

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

```
which(sapply(requisitos, "%in%", stock))
```

```
##   van carro carro
##     3     5     7
```

Más datos sobre búsqueda de patrones

Para nuestros propósitos de especificar estos patrones, ciertos caracteres son llamados **metacaracteres**, especificamente `\ | ( ) [ ] { } ^ $ * + ?`, así cualquier metacaracter con significado especial en nuestro string puede ser citado precediéndolo con un backslash `\\`, es decir si quiero el significado literal de `{` usamos `\{`.

Existen varias tipos de sintaxis par ala búsqueda de patrones, sin embargo, en R, las expresiones regulares son especificadas por **POSIX (POrtable Operating System Interface** 1003.2, ya sea extendido o básico, dependiendo sobre el valor del argumento extendido, a menos que `perl=TRUE`, lo que corresponde a que usaremos las expresiones regulares tipo PCRE (Perl Compatible Regular Expresions)

Observe que las llaves cuadradas [] son parte de los nombres simbólicos y deben ser incluídos en adición a las llave, por ejemplo [[:alnum:]] significa [0-9A-Za-z], algunas interpretaciones de POSIX las vemos en el documento adjunto, por ejemplo `\{n\}` queiere decir el patrón es buscado n o más veces, `\{,m\}` el patrón es buscado hasta m veces, `\{m,n}\` el patrón es buscado entre n y m veces

```
text <- c("arm","leg","head", "foot","hand", "hindleg", "elbow")
```

```
grep("o{1}",text,value=T)
```

```
## [1] "foot"  "elbow"
```

```
grep("o{2}",text,value=T)
```

```
## [1] "foot"
```

```
grep("o{3}",text,value=T)
```

```
## character(0)
```

```
grep("[[:alnum:]]{4, }",text,value=T)
```

```
## [1] "head"    "foot"    "hand"    "hindleg" "elbow"
```

```
grep("[[:alnum:]]{5, }",text,value=T)
```

```
## [1] "hindleg" "elbow"
```

```
grep("[[:alnum:]]{6, }",text,value=T)
```

```
## [1] "hindleg"
```

```
grep("[[:alnum:]]{7, }",text,value=T)
```

```
## [1] "hindleg"
```

Eliminar texto estampado en cadenas complejas

Suponga que queremos separar la información en estas cadenas complicadas

```
entradas <- c("Prueba 1 58 cervoconis (52 coincidencias)","Prueba 2 60 terrestris (51 conincidencias)",
entradas
```

```
## [1] "Prueba 1 58 cervoconis (52 coincidencias)"
## [2] "Prueba 2 60 terrestris (51 conincidencias)"
## [3] "Prueba 8 109 flavicollis (101 coincidencias)"
```

la primera tarea es remover el material del número de coincidencias

```
gsub("\\(.*\\)$", "", entradas)
```

```
## [1] "Prueba 1 58 cervoconis "   "Prueba 2 60 terrestris "
## [3] "Prueba 8 109 flavicollis "
```

```
gsub(" *$", "",gsub("\\(.*\\)$", "", entradas))
```

```
## [1] "Prueba 1 58 cervoconis"   "Prueba 2 60 terrestris"
## [3] "Prueba 8 109 flavicollis"
```

```
pos <- regexpr("\\(.*\\)$", entradas)
pos
```

```
## [1] 24 24 26
## attr(,"match.length")
## [1] 18 19 19
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

```
substring(entradas, first=pos+1, last=pos+attr(pos,"match.length")-2)
```

```
## [1] "52 coincidencias"  "51 conincidencias" "101 coincidencias"
```

Fechas y tiempos en R

El manejo de datos se vuelve altamente complejo dado la variabilidad que tiene este, veamos como R posee un sistema robusto que trata con esta complejidad. Para ver como R maneja fechas y horas, echemos una mirada en

```
Sys.time()
```

```
## [1] "2020-05-02 14:16:27 -05"
```

cabe aclarar que la zona horaria e despliega en string, esta represetación de la fecha y hora como un caracter es bastante amigable pero no es buena para cálculo, con lo que necesitamos una expresión numérica para combinar fechas y horas. La convención que R utiliza está basada en segundos. Usted puede siempre agregar hasta días o años, pero no puede hacer lo contrario. La línea de base para expresar la fecha y hora de hoy en segundos es el 1 de enero de 1970:

```
as.numeric(Sys.time())
```

```
## [1] 1588446988
```

Esto está bien para graficar series de tiempo, pero no para calcular que mes corresponde o qué día. Para responder estas cuestiones podemos acceder al tablero de variables categoricas asociadas con la fecha. Para acomodar esto R, usa el sistema POSIX para representar tiempos y fechas

```
class(Sys.time())
```

```
## [1] "POSIXct" "POSIXt"
```

Podemos pensar en la clase POSIXct con el sufijo ct continuo en el tienmpo y POSIXlt con el sufijo lista de tiempo

```
time.list <- as.POSIXlt(Sys.time())
unlist(time.list)
```

```
##                sec                min               hour               mday
## "27.9386761188507"              "16"              "14"               "2"
##                mon               year               wday               yday
##                "4"              "120"               "6"              "122"
##              isdst               zone             gmtoff
##                "0"              "-05"          "-18000"
```

Aquí la hora está en formato militar, el mes con el correpondiente número mday es el día del mes, wday es el día de la semana con domingo=0 a sábado=6, el día del año con yday, year es el año teniendo en cuenta 0=1900 e isdst es una variable lógica si está en horario de verano

Leyendo tiempos de un archivo

Es muy probable que sus datos estén en formato Excel, por ejemplo 03/09/2014, donde día/mes/año, así

```
setwd("~/Progrmación en R/2020-I/PR06-caracteres y datos de tiempo")
data <- read.table("dates.txt",header=T)
attach(data)
data
```

```
##   x       date
## 1 3 15/06/2014
## 2 1 16/06/2014
## 3 6 17/06/2014
## 4 7 18/06/2014
```

```
## 5 8 19/06/2014
## 6 9 20/06/2014
```

cuando se usa `read.table()` este por defecto toma los valore de fecha y los convierte de tipo `factor`

```
mode(date)
```

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

```
class(date)
```

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

El punto es que para nuestros porpósitos, R de inicio no reconoce las fechas como tipo `date`. para esto empleamos la función `strptime`

Para esta función proveemos un formato establecido en comillas en el que le decimos a R que esperar exactamente, en qué orden y separado bajo qué símbolos. Para este ejemplo tenemos días (en 2 dígitos), meses y años, separados por slashes

```
Rdate <- strptime(as.character(date),"%d/%m/%Y")
Rdate
```

```
## [1] "2014-06-15 -05" "2014-06-16 -05" "2014-06-17 -05" "2014-06-18 -05"
## [5] "2014-06-19 -05" "2014-06-20 -05"
```

```
class(Rdate)
```

```
## [1] "POSIXlt" "POSIXt"
```

Con lo que simepre es buena idea poner los datos en el data frame

```
data <- data.frame(data,Rdate)
data
```

```
##   x       date      Rdate
## 1 3 15/06/2014 2014-06-15
## 2 1 16/06/2014 2014-06-16
## 3 6 17/06/2014 2014-06-17
## 4 7 18/06/2014 2014-06-18
## 5 8 19/06/2014 2014-06-19
## 6 9 20/06/2014 2014-06-20
```

Con estos datos podemos hacer cálculos para saber por ejemplo el valor medio de x de cada día de la semana

```
tapply(x,Rdate$wday,mean)
```

```
## 0 1 2 3 4 5
## 3 1 6 7 8 9
```

así la menor media es los lunes y la mayor los viernes, es comlicado recordar los formatos de strptime, equí los recordamos

poner la imagen

Existe una función útil que nos despliega el nombre de la semana

```
y <- strptime("01/02/2014",format="%d/%m/%Y")
weekdays(y)
```

```
## [1] "sábado"
```

el cual fue convertido de

```
y$wday
```

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

Otro tipo de fecha, tiene que ver con los años en 2 dígitos (`%y`), y los meses abreviados (`%b`) si usar separadores

```
other.dates <- c("1ene.99", "2ene.05", "31mar.04", "30jul.05")
strptime(other.dates, "%d%b%y")
```

```
## [1] "1999-01-01 -05" "2005-01-02 -05" "2004-03-31 -05" "2005-07-30 -05"
```

Aquí otra posibilidad con el año, mes con el nombre completo y, seguido de la semana del año, y el día de la semana abreviadas

```
yet.another.date <- c("2016 enero 2 lun.","2017 febrero 6 vie.","2018
marzo 10 jue.")
strptime(yet.another.date,"%Y %B %W %a")
```

```
## [1] "2016-01-11 -05" "2017-02-10 -05" "2018-03-08 -05"
```

el sistema es inteligente en saber la fecha del lunes en la semana número 2 de enero de 2016

```
yet.more.dates <- c("2016 2 lun.","2017 6 vie.","2018 10 jue.")
strptime(yet.more.dates,"%Y %W %a")
```

```
## [1] "2016-01-11 -05" "2017-02-10 -05" "2018-03-08 -05"
```

Función difftime

La función `difftime` la diferencia entre 2 objetos de fechas en el tiempo y retorna un objeto de clase `difftime` con un atributo indicando la unidad, se puede usar varias operaciones aritméticas sobre `difftime` como `round`,`signif`,`floor`,`ceiling`,`trunc`,`abs`,`sign` y ciertos operadores lógicos. Podemos crear un objeto `difftime` de la siguiente manera

```
yet.more.dates <- c("2016 2 lun.","2017 6 vie.","2018 10 jue.")
as.difftime(yet.more.dates,"%Y %W %a")
```

```
## Time differences in days
## [1] -1573 -1177  -786
```

o

```
difftime("2014-02-06","2014-07-06")
```

```
## Time difference of -150 days
```

```
round(difftime("2014-02-06","2014-07-06"),0)
```

```
## Time difference of -150 days
```

Cálculos con fechas y tiempo

Usted puede hacer las siguientes operaciones con fechas y tiempos

- tiempo + número
- tiempo - número
- tiempo1 - tiempo2

- tiempo1 operación lógica tiempo2

donde los operadores lógicos son `==`,`!=`,`<`,`<=`,`>` o `>=` usted puede agragar o sustraer un número de segundos o un objeto `difftime` desde un objeto de fecha-tiempo, pero no puede agregar dos objetos fecha-tiempo. LA sustracción de dos objetos fecha-tiempo es equivalente a usar `difftime`. A menos que la zona horario haya sido especificada,los objetos tipo `POSIXlt` son interpetados estando en la zona horaria actual en los cálculo

Lo que debe comprender es que debe convertir sus fechas y horas en objetos POSIXlt antes de comenzar a hacer cualquier cálculo. Una vez que son objetos POSIXlt, es sencillo calcular medias, diferencias, etc. Aquí queremos calcular el número de días entre dos fechas, 22 de octubre de 2015 y 22 de octubre de 2018:

```
y2 <- as.POSIXlt("2015-10-22")
y1 <- as.POSIXlt("2018-10-22")
y1-y2
```

```
## Time difference of 1096 days
```

Las funciones difftime y as.difftime

Resolver la diferencia horaria entre dos fechas y horas involucra la función `difftime`, que toma dos objetos de fecha y hora como argumentos. La función devuelve un objeto de clase `difftime` con un atributo que indica las unidades. Por ejemplo, ¿cuántos días transcurrieron entre el 15 de agosto de 2013 y el 21 de octubre de 2015?

```
difftime("2015-10-21","2013-8-15")
```

```
## Time difference of 797 days
```

si usted quiere el número de días para usar en el cálculo

```
as.numeric(difftime("2015-10-21","2013-8-15"))
```

```
## [1] 797
```

Su usted tiene tiempo pero no fechas, entonces usted puede usar la función `as.difftime` para crear objetos apropiados para los cálculos

```
t1 <- as.difftime("6:14:21")
t2 <- as.difftime("5:12:32")
t1-t2
```

```
## Time difference of 1.030278 hours
```

A menudo querrá crear objetos POSIXlt a partir de componentes almacenados en diferentes vectores dentro de un dataframe. Por ejemplo, aquí hay un dataframe con las horas, minutos y segundos de un experimento con dos niveles de factores en cuatro columnas separadas:

```
setwd("~/Progrmación en R/2020-I/PR06-caracteres y datos de tiempo")
times <- read.table("times.txt",header=T)
attach(times)
times
```

```
##    hrs min sec experiment
## 1    2  23   6          A
## 2    3  16  17          A
## 3    3   2  56          A
## 4    2  45   0          A
## 5    3   4  42          A
## 6    2  56  25          A
## 7    3  12  28          A
```

```
## 8    1  57  12          A
## 9    2  22  22          B
## 10   1  42   7          B
## 11   2  31  17          B
## 12   3  15  16          B
## 13   2  28   4          B
## 14   1  55  34          B
## 15   2  17   7          B
## 16   1  48  48          B
```

Peguemos los valores dado que los tiempo no están en formato POSIXlt

```
paste(hrs,min,sec,sep=":")
```

```
##  [1] "2:23:6"  "3:16:17" "3:2:56"  "2:45:0"  "3:4:42"  "2:56:25" "3:12:28"
##  [8] "1:57:12" "2:22:22" "1:42:7"  "2:31:17" "3:15:16" "2:28:4"  "1:55:34"
## [15] "2:17:7"  "1:48:48"
```

ahora guardemos las diferencias en un vector de `difftime`

```
duracion <- as.difftime (paste(hrs,min,sec,sep=":"))
duracion
```

```
## Time differences in hours
##  [1] 2.385000 3.271389 3.048889 2.750000 3.078333 2.940278 3.207778 1.953333
##  [9] 2.372778 1.701944 2.521389 3.254444 2.467778 1.926111 2.285278 1.813333
```

llevemos a cabo nuestros cálculos como media varianza usando `tapply`

```
tapply(duracion,experiment,mean)
```

```
##        A        B
## 2.829375 2.292882
```

cuya respuesta está en horas decimales

Generacción de secuencias en el tiempo

Podemos generar secuencias de fechas por años, meses, semanas días del mes o días de la semana, veamos nuestro primer ejemplo incrementando en un día

```
seq(as.POSIXlt("2015-11-04"), as.POSIXlt("2015-11-15"), "1 day")
```

```
##  [1] "2015-11-04 -05" "2015-11-05 -05" "2015-11-06 -05" "2015-11-07 -05"
##  [5] "2015-11-08 -05" "2015-11-09 -05" "2015-11-10 -05" "2015-11-11 -05"
##  [9] "2015-11-12 -05" "2015-11-13 -05" "2015-11-14 -05" "2015-11-15 -05"
```

incrementando en 2 semanas

```
seq(as.POSIXlt("2015-11-04"), as.POSIXlt("2016-04-05"), "2 weeks")
```

```
##  [1] "2015-11-04 -05" "2015-11-18 -05" "2015-12-02 -05" "2015-12-16 -05"
##  [5] "2015-12-30 -05" "2016-01-13 -05" "2016-01-27 -05" "2016-02-10 -05"
##  [9] "2016-02-24 -05" "2016-03-09 -05" "2016-03-23 -05"
```

en incrementos de 3 meses

```
seq(as.POSIXlt("2015-11-04"), as.POSIXlt("2018-10-04"), "3 months")
```

```
##  [1] "2015-11-04 -05" "2016-02-04 -05" "2016-05-04 -05" "2016-08-04 -05"
##  [5] "2016-11-04 -05" "2017-02-04 -05" "2017-05-04 -05" "2017-08-04 -05"
```

```
##  [9] "2017-11-04 -05" "2018-02-04 -05" "2018-05-04 -05" "2018-08-04 -05"
```

con incremento de un año

```
seq(as.POSIXlt("2015-11-04"), as.POSIXlt("2026-02-04"), "year")
```

```
##  [1] "2015-11-04 -05" "2016-11-04 -05" "2017-11-04 -05" "2018-11-04 -05"
##  [5] "2019-11-04 -05" "2020-11-04 -05" "2021-11-04 -05" "2022-11-04 -05"
##  [9] "2023-11-04 -05" "2024-11-04 -05" "2025-11-04 -05"
```

Si especifica un número, en lugar de una cadena de caracteres reconocida, en la parte de la función de secuencia, entonces se supone que el número es un número de segundos, por lo que esto genera la hora y la fecha:

```
seq(as.POSIXlt("2015-11-04"), as.POSIXlt("2015-11-05"), 8955)
```

```
##  [1] "2015-11-04 00:00:00 -05" "2015-11-04 02:29:15 -05"
##  [3] "2015-11-04 04:58:30 -05" "2015-11-04 07:27:45 -05"
##  [5] "2015-11-04 09:57:00 -05" "2015-11-04 12:26:15 -05"
##  [7] "2015-11-04 14:55:30 -05" "2015-11-04 17:24:45 -05"
##  [9] "2015-11-04 19:54:00 -05" "2015-11-04 22:23:15 -05"
```

Al igual que con otras formas de seq, puede especificar la longitud del vector que se generará, en lugar de especificar la fecha final:

```
seq(as.POSIXlt("2015-11-04"), by="month", length=10)
```

```
##  [1] "2015-11-04 -05" "2015-12-04 -05" "2016-01-04 -05" "2016-02-04 -05"
##  [5] "2016-03-04 -05" "2016-04-04 -05" "2016-05-04 -05" "2016-06-04 -05"
##  [9] "2016-07-04 -05" "2016-08-04 -05"
```

o puede generar un vector de fechas para que coincida con la longitud de un vector existente, utilizando `along =` en lugar de `length =`

```
results <- runif(16)
seq(as.POSIXlt("2015-11-04"), by="month", along=results )
```

```
##  [1] "2015-11-04 -05" "2015-12-04 -05" "2016-01-04 -05" "2016-02-04 -05"
##  [5] "2016-03-04 -05" "2016-04-04 -05" "2016-05-04 -05" "2016-06-04 -05"
##  [9] "2016-07-04 -05" "2016-08-04 -05" "2016-09-04 -05" "2016-10-04 -05"
## [13] "2016-11-04 -05" "2016-12-04 -05" "2017-01-04 -05" "2017-02-04 -05"
```

Usted puede usar `weekdays` para extraer los días de la semana de una serie de datos

```
weekdays(seq(as.POSIXlt("2015-11-04"), by="month", along=results ))
```

```
##  [1] "miércoles" "viernes"   "lunes"     "jueves"    "viernes"   "lunes"
##  [7] "miércoles" "sábado"    "lunes"     "jueves"    "domingo"   "martes"
## [13] "viernes"   "domingo"   "miércoles" "sábado"
```

Suponga que desea encontrar las fechas de todos los lunes en una secuencia de fechas. Esto implica el uso de subíndices lógicos. Se seleccionarán los subíndices que evalúen como VERDADERO, por lo que la declaración lógica que debe realizar es `wday == 1`. (porque el domingo es `wday == 0`). Cree un objeto llamado `y` que contenga los primeros 100 días en 2016 (tenga en cuenta que la fecha de inicio es el 31 de diciembre de 2015), luego convierta este vector de fechas en un objeto POSIXlt, una lista llamada `x`, como esta:

```
y <- as.Date(1:100,origin="2015-12-31")
y
```

```
##   [1] "2016-01-01" "2016-01-02" "2016-01-03" "2016-01-04" "2016-01-05"
##   [6] "2016-01-06" "2016-01-07" "2016-01-08" "2016-01-09" "2016-01-10"
```

```
##  [11] "2016-01-11" "2016-01-12" "2016-01-13" "2016-01-14" "2016-01-15"
##  [16] "2016-01-16" "2016-01-17" "2016-01-18" "2016-01-19" "2016-01-20"
##  [21] "2016-01-21" "2016-01-22" "2016-01-23" "2016-01-24" "2016-01-25"
##  [26] "2016-01-26" "2016-01-27" "2016-01-28" "2016-01-29" "2016-01-30"
##  [31] "2016-01-31" "2016-02-01" "2016-02-02" "2016-02-03" "2016-02-04"
##  [36] "2016-02-05" "2016-02-06" "2016-02-07" "2016-02-08" "2016-02-09"
##  [41] "2016-02-10" "2016-02-11" "2016-02-12" "2016-02-13" "2016-02-14"
##  [46] "2016-02-15" "2016-02-16" "2016-02-17" "2016-02-18" "2016-02-19"
##  [51] "2016-02-20" "2016-02-21" "2016-02-22" "2016-02-23" "2016-02-24"
##  [56] "2016-02-25" "2016-02-26" "2016-02-27" "2016-02-28" "2016-02-29"
##  [61] "2016-03-01" "2016-03-02" "2016-03-03" "2016-03-04" "2016-03-05"
##  [66] "2016-03-06" "2016-03-07" "2016-03-08" "2016-03-09" "2016-03-10"
##  [71] "2016-03-11" "2016-03-12" "2016-03-13" "2016-03-14" "2016-03-15"
##  [76] "2016-03-16" "2016-03-17" "2016-03-18" "2016-03-19" "2016-03-20"
##  [81] "2016-03-21" "2016-03-22" "2016-03-23" "2016-03-24" "2016-03-25"
##  [86] "2016-03-26" "2016-03-27" "2016-03-28" "2016-03-29" "2016-03-30"
##  [91] "2016-03-31" "2016-04-01" "2016-04-02" "2016-04-03" "2016-04-04"
##  [96] "2016-04-05" "2016-04-06" "2016-04-07" "2016-04-08" "2016-04-09"
```

```
x <- as.POSIXlt(y)
```

Ahora, dado que `x` es una lista, puede usar el operador `$` para acceder a la información en días laborables y, por supuesto, descubre que están separados por 7 días, a partir del 4 de enero de 2016:

```
x[x$wday==1]
```

```
##  [1] "2016-01-04 UTC" "2016-01-11 UTC" "2016-01-18 UTC" "2016-01-25 UTC"
##  [5] "2016-02-01 UTC" "2016-02-08 UTC" "2016-02-15 UTC" "2016-02-22 UTC"
##  [9] "2016-02-29 UTC" "2016-03-07 UTC" "2016-03-14 UTC" "2016-03-21 UTC"
## [13] "2016-03-28 UTC" "2016-04-04 UTC"
```

Suponga que desea enumerar las fechas del primer lunes de cada mes. Esta es la fecha con `wday == 1` (como arriba) pero solo en su primera aparición en cada mes del año. Esto es un poco más complicado, porque varios meses contendrán cinco lunes, por lo que no puede usar `seq` con `by = "28 days"` para resolver el problema (esto generaría 13 fechas, no las 12 requeridas). Aquí están las fechas de todos los lunes del año 2016:

```
y <- as.POSIXlt(as.Date(1:365,origin="2015-12-31"))
```

Esto es lo que sabemos hasta ahora:

```
data.frame(monday=y[y$wday==1],month=y$mo[y$wday==1])[1:12,]
```

```
##        monday month
## 1  2016-01-04     0
## 2  2016-01-11     0
## 3  2016-01-18     0
## 4  2016-01-25     0
## 5  2016-02-01     1
## 6  2016-02-08     1
## 7  2016-02-15     1
## 8  2016-02-22     1
## 9  2016-02-29     1
## 10 2016-03-07     2
## 11 2016-03-14     2
## 12 2016-03-21     2
```

Desea que un vector marque los 12 lunes que necesita: estos son aquellos en los que el mes no está duplicado (es decir, desea tomar la primera fila de cada mes). Para este ejemplo, el primer lunes de enero está en la

fila 1 (obviamente), el primero en febrero en la fila 5, el primero en marzo en la fila 10, y así sucesivamente. ¡Puede usar la función no duplicada! Duplicada para etiquetar estas filas

```
wanted <- !duplicated(y$mo[y$wday==1])
```

Por último, seleccione las 12 fechas de los primeros lunes usando quería como subíndice como este:

```
y[y$wday==1][wanted]
```

```
##  [1] "2016-01-04 UTC" "2016-02-01 UTC" "2016-03-07 UTC" "2016-04-04 UTC"
##  [5] "2016-05-02 UTC" "2016-06-06 UTC" "2016-07-04 UTC" "2016-08-01 UTC"
##  [9] "2016-09-05 UTC" "2016-10-03 UTC" "2016-11-07 UTC" "2016-12-05 UTC"
```

Tenga en cuenta que cada mes está representado, y ninguna de las fechas es posterior al 7 del mes, según sea necesario.

Cálculo de diferencias entre las filas de un dataframe

Una acción común con datos de tiempo es calcular la diferencia de tiempo entre filas sucesivas de un dataframe. El vector llamado duracion creado anteriormente es de clase `difftime` y contiene 16 veces medido en horas decimales:

```
class(duracion)
```

```
## [1] "difftime"
```

```
duracion
```

```
## Time differences in hours
##  [1] 2.385000 3.271389 3.048889 2.750000 3.078333 2.940278 3.207778 1.953333
##  [9] 2.372778 1.701944 2.521389 3.254444 2.467778 1.926111 2.285278 1.813333
```

Puede calcular las diferencias entre filas sucesivas utilizando subíndices, como este:

```
duracion[1:15]-duracion[2:16]
```

```
## Time differences in hours
##  [1] -0.8863889  0.2225000  0.2988889 -0.3283333  0.1380556 -0.2675000
##  [7]  1.2544444 -0.4194444  0.6708333 -0.8194444 -0.7330556  0.7866667
## [13]  0.5416667 -0.3591667  0.4719444
```