Programación en R

Grupo Tortillas Ninja

1° Semestre 2021

Consigas del trabajo

El trabajo que hay que presentar implica revisar los algoritmos que se presentan a continuación. Deberá ejecutarlos primero en la línea de comando de la consola.

Luego deberá elegir alguno de los métodos vistos para medir la performance y comparar los resultados con otros compañeros que hayan usado otros métodos para medir la performance.

Luego todo deberá entregarse en un informe en formato pdf construido con RStudio, archivo RMarkdow.

Algoritmos a revisar y Metodos para medir la performance

Los algoritmos a revisar son:

- Generacion de un *Vector Secuencia*,
- Implementacion de una *Serie de Fibonacci*,
- Ordenacion de un vector por *Metodo burbuja*.

Los metodos para ver la performance son:

- Sys.time,
- Biblioteca tictoc,
- Biblioteca rbenchmark,
- Biblioteca Microbenchamrk.

Algoritmo n°1: Generacion de un Vector Secuencia

Al leer el archivo dado por la catedra, vemos que disponemos de 2 maneras de generar un vector secuencia:

1. Usando el bucle *for*,

Debemos generar una secuencia de numero entre 0 y 100000, que vaya de 2 en 2.

Primero ejecutamos en la consola de la maquina para comprender el funcionamiento. Luego de hacer esto, vemos las lineas de codigo abajo:

```
vec1 <- vector()
for(var1 in 0:50000){
  vec1[var1+1] <- var1*2
}
tail(vec1)
```

```
## [1]  99990  99992  99994  99996  99998 100000
```

```
head(vec1)
```

```
## [1]  0  2  4  6  8 10
```

El uso de los comandos tail y head es para visualizar el principío (head/cabeza) y el final (tail/cola) del vector secuencia y comprobar que esta hecho correctamente.

2. Usando el comando de R, *seq*.

Analogamente, primero ejecutamos en la consola de la maquina para comprender el funcionamiento. Luego de hacer esto, vemos las lineas de codigo abajo:

```
vec2 <- vector()
vec2 <- seq(0,100000,2)
tail(vec2)
```

```
## [1]  99990  99992  99994  99996  99998 100000
```

```
head(vec2)
```

```
## [1]  0  2  4  6  8 10
```

El uso de los comandos tail y head es para visualizar el principío (head/cabeza) y el final (tail/cola) del vector secuencia y comprobar que esta hecho correctamente.

Medicion de la performance

Paso siguiente, deberemos comparar la perfromance de ambos con alguno de los metodos anterioremnte mencionados. En este caso elegimos usar la *Biblioteca tictoc*. Es imporante mencionar que medimos la performance en costo computacional, lo cual lo vemos en el tiempo de ejecucion.

En la seccion de Ayuda de RStudio vemos que define lo que hacen estos comando de la siguiente manera: *"In general, calls to tic and toc start the timer when the tic call is made and stop the timer when the toc call is made, recording the elapsed time between the calls from proc.time."*

A continuacion vemos el algoritmo que va a medir los tiempos:

```
library(tictoc)
vec1 <- vector()
vec2 <- vector()

tic.clearlog()

tic("Tiempo de ejecucion bucle for: ")
for(var1 in 0:50000){
  vec1[var1+1] <- var1*2
}
toc(log = TRUE, quiet = TRUE)

tic("Tiempo de ejecucion comando seq: ")
vec2 <- seq(0,100000,2)
toc(log = TRUE, quiet = TRUE)

unlist(tic.log())
```

```
## [1] "Tiempo de ejecucion bucle for: : 0.01 sec elapsed"
## [2] "Tiempo de ejecucion comando seq: : 0.02 sec elapsed"
```

Algoritmo n°2: Implementacion de una Serie de Fibonacci

En matemáticas, la sucesión o serie de Fibonacci es la sucesión infinita de numeros naturales que se crea a partir de los numeros 0, 1 y 1, y luego de ellos cada termino es la suma de los dos anteriores. Esto nos

da:0,1,1,2,3,5,8 . . . 89,144,233 . . .

A continuacion estan las lineas de codigo para armar los primeros 20 valores de la Sucesion de Fibonacci:

```
w <- vector()
w[1]<- 0
w[2]<- 1
for(var in 0:17)
{
  w[var+3] <- w[var+2]+w[var+1]
}
print(w)
```

```
##  [1]    0    1    1    2    3    5    8   13   21   34   55   89  144  233  377
## [16]  610  987 1597 2584 4181
```

Medicion de la performance

Luego, deberemos comparar la perfromance con alguno de los metodos anterioremnte mencionados. En este caso elegimos usar *Biblioteca rbenchmark*.

En la internet vemos que hace este paquete: *"The library consists of just one function, benchmark, which is a simple wrapper around system.time. Given a specification of the benchmarking process (counts of replications, evaluation environment) and an arbitrary number of expressions, benchmark evaluates each of the expressions in the specified environment, replicating the evaluation as many times as specified, and returning the results conveniently wrapped into a data frame"*

En las siguientes lineas de codigo vemos como medimos la performance de un programa que evalua los primeros 200 terminos de la Serie de Fibonacci, repitiendo el proceso 1000 veces:

```
library(rbenchmark)

k <- vector()
w <- vector()

w = function(it){
  k[1] <- 0
  k[2] <- 1
  for(i in 0:it){
    k[i+3] <- (k[i+2]+k[i+1])
  }
  return(k)
}


it=197
benchmark(w(it), replications = 1000)
```

```
##    test replications elapsed relative user.self sys.self user.child sys.child
## 1 w(it)         1000    0.08        1      0.04     0.03         NA        NA
```

En la tabla vemos:

- *elapsed*: nos muestra el tiempo acumulado en todas las 1000 replicaciones.
- *relative*: nos muestra la razon con la prueba mas rapida.
- *user.self*: gives the CPU time spent by the current process.
- *sys.self*: gives the CPU time spent by the kernel (the operating system) on behalf of the current process.

Algoritmo n°3: Ordenacion de un vector por Metodo Burbuja

La Ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

A continuacion estan las lineas de codigo para ordenar, de mayor a menor, por metodo burbuja un vector de 150 numeros elegidos aleatoriamente entre el 1 y el 1000000:

```
muestra <- sample(1:1000000,150)

head(muestra)
```

```
## [1] 145497 911008  23812 552112 341930 953434
```

```
tail(muestra)
```

```
## [1] 957447 567089 461520 585721 651501 177945
```

```
burbuja = function(muestra){
  n <- length(muestra)
  for(i in 1:(n-1)){
    for(j in 1:(n-i)){
      if (muestra[j] < muestra[j+1]){
        temporal <- muestra[j]
        muestra[j] <- muestra[j+1]
        muestra[j+1] <- temporal
      }
    }
  }
  return(muestra)
}

muestra_ordenada <- burbuja(muestra)

head(muestra_ordenada)
```

```
## [1] 989477 977735 977499 975209 964067 964030
```

```
tail(muestra_ordenada)
```

```
## [1] 45678 41840 38811 24408 23812  4737
```

Medicion de la performance

Luego, deberemos comparar la perfromance con alguno de los metodos anterioremnte mencionados. En este caso elegimos usar *Biblioteca Microbenchmark*.

En internet encontramos esto sobre este metodo: *"Microbenchmark serves as a more accurate replacement of the often seen system.time(replicate(1000, expr)) expression. It tries hard to accurately measure only the time it takes to evaluate expr. To achieved this, the sub-millisecond (supposedly nanosecond) accurate timing functions most modern operating systems provide are used."*

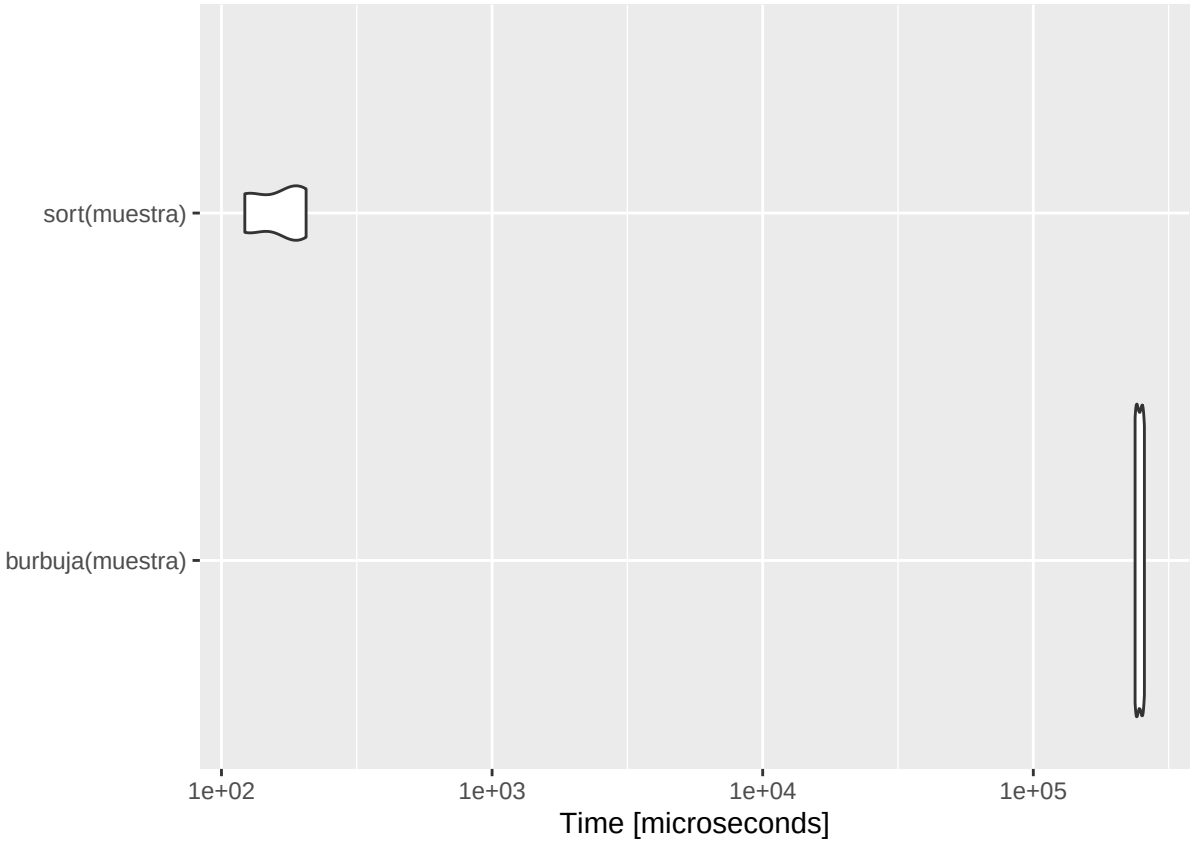
A continuacion vemos las lineas de codigo para calcular el tiempo de ejecucion de el codigo para ordenar por metodo burbuja. Tomaremos una muestra de 2000 numeros entre el 1 y el 1000000 Lo compararemos con el comando *sort* de R:

```r
library(microbenchmark)

muestra <- sample(1:1000000,2000)

burbuja = function(muestra){
  n <- length(muestra)
  for(i in 1:(n-1)){
    for(j in 1:(n-i)){
      if (muestra[j] < muestra[j+1]){
        temporal <- muestra[j]
        muestra[j] <- muestra[j+1]
        muestra[j+1] <- temporal
      }
    }
  }
  return(muestra)
}


microbenchmark(burbuja(muestra),sort(muestra), times=5)
```

```
## Unit: microseconds
##              expr      min       lq      mean   median       uq      max neval
##  burbuja(muestra) 238579.7 239835.0 243378.48 243185.4 245125.3 250167.0     5
##     sort(muestra)    123.1    188.6    204.54    191.5    194.9    324.6     5
```

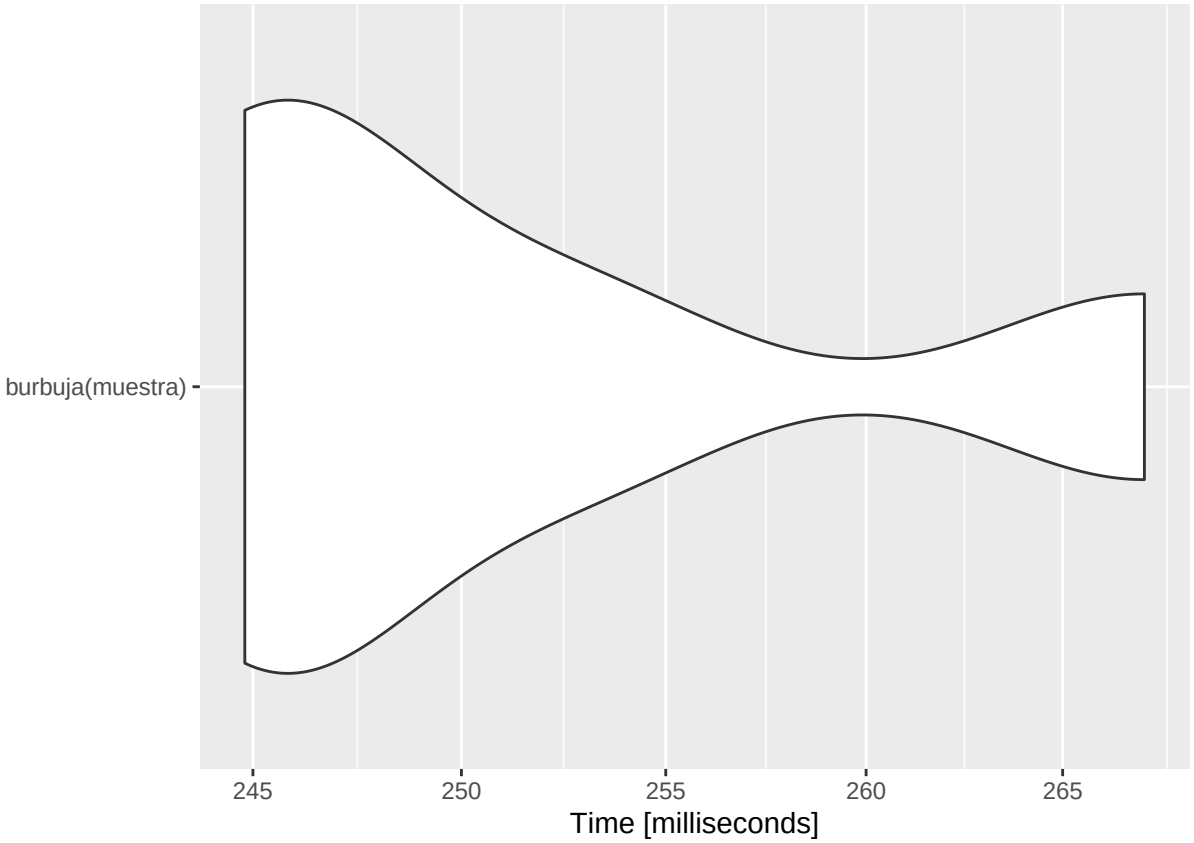
```r
mbm <- microbenchmark(burbuja(muestra),sort(muestra), times=5)

library(ggplot2)
autoplot(mbm)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



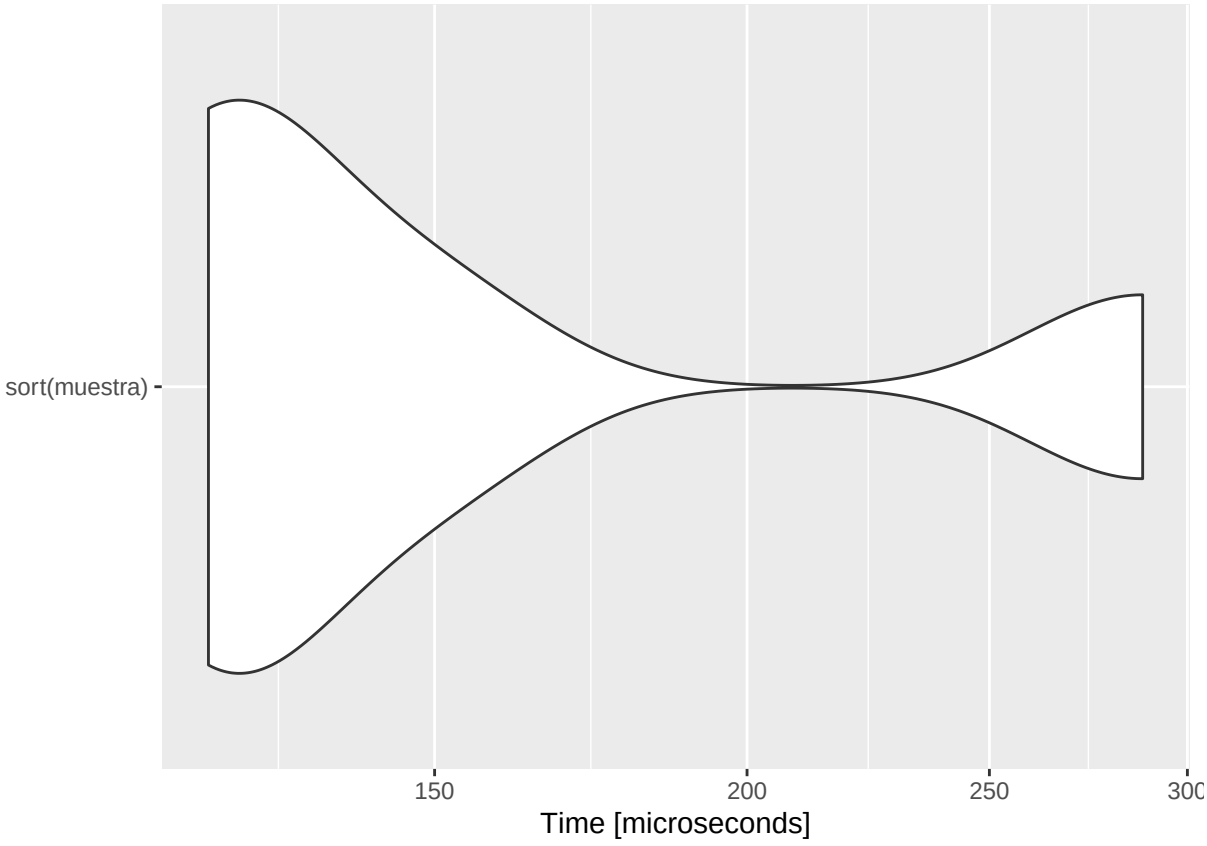

```
autoplot(microbenchmark(burbuja(muestra),times=5))
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```




```
autoplot(microbenchmark(sort(muestra),times=5))
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



Caso de aplicación Rpub - Calentamiento global: un panel de visualización de datos

Introducción

¡Hola! Gracias por tomarse su tiempo para ver este panel.

Este tablero presenta información de diversas fuentes sobre diferentes aspectos del calentamiento global.

Primero, los espectadores pueden ver una representación directa del calentamiento global en sí, que se muestra en el aumento de la temperatura global. Luego, se presentan visualizaciones de la emisión de gases de efecto invernadero, que es la principal causa del calentamiento global. Finalmente, se incluyen un par de efectos del calentamiento global.

El panel utiliza datos de código abierto de varias fuentes, como:

*Berkeley Earth

*Gapminder

*Datahub

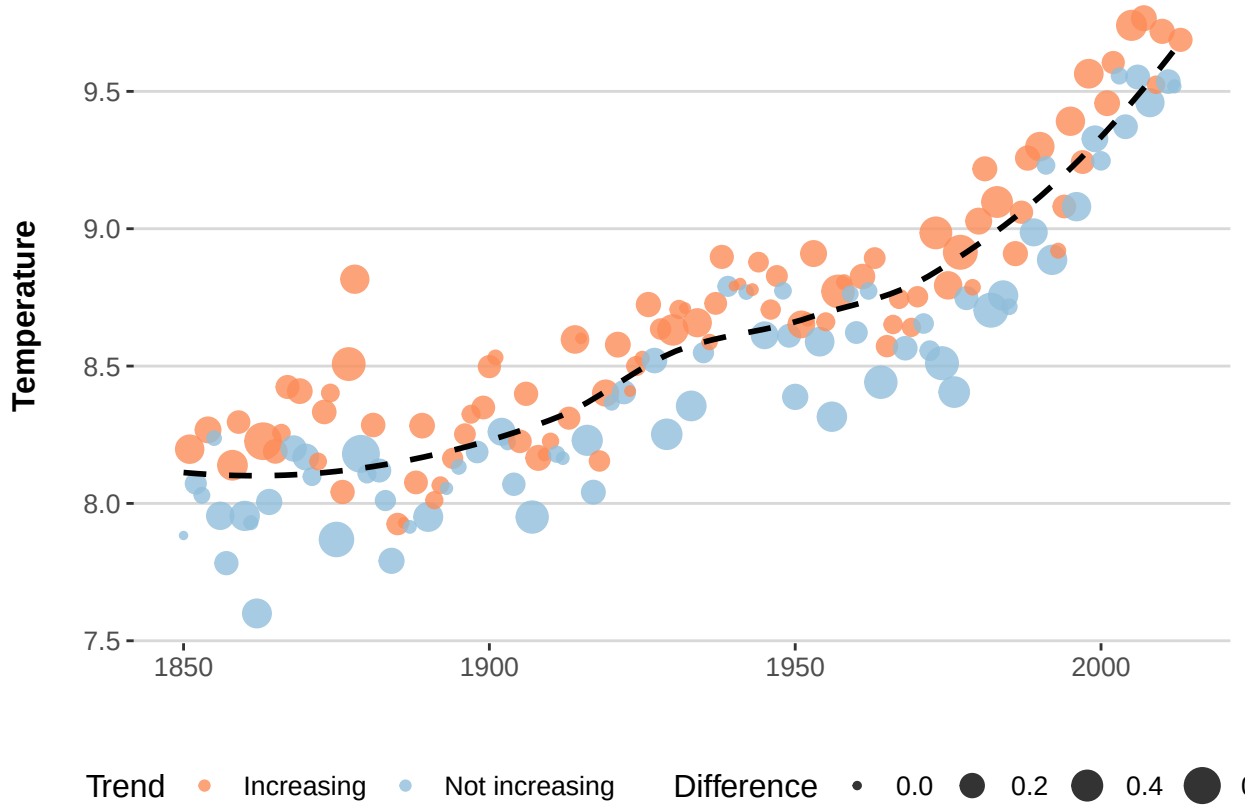
*World Bank Open Data

Los códigos utilizados para este panel, incluidos los utilizados en la negociación de datos y los gráficos individuales, se pueden encontrar en el sigueinte [repositorio de GitHub] (https://github.com/Gianatmaja/Global-Warming-Dashboard).

Temperatura global promedio de la tierra (1850-2013)

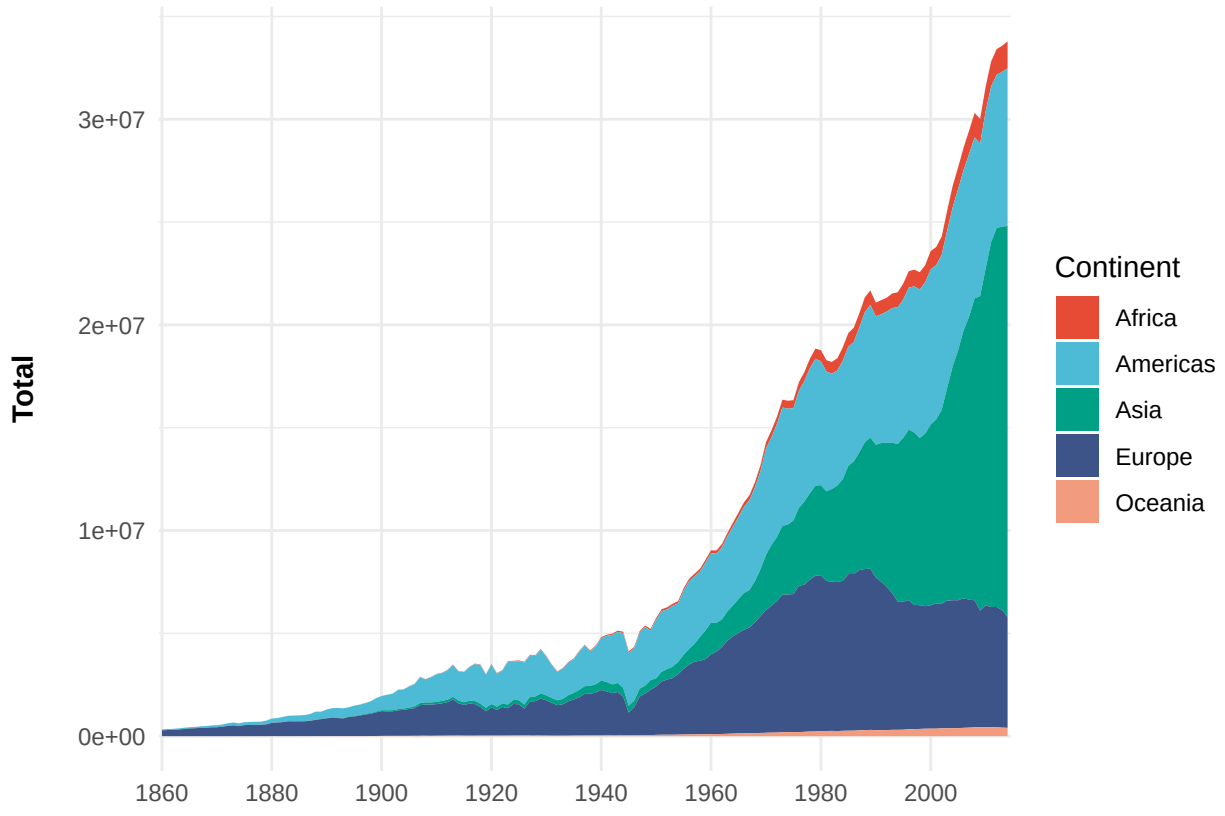
```
P1
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



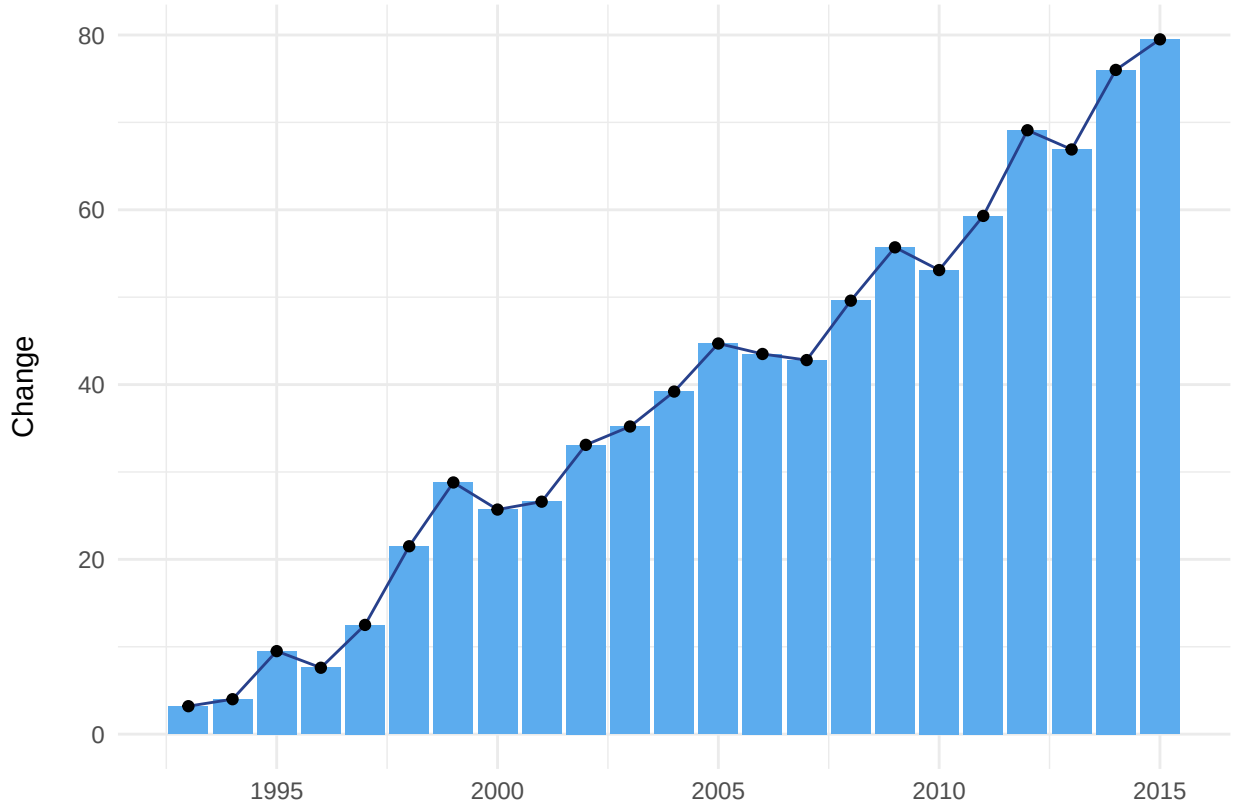
Emisiones de CO2 (1860-2014)

```
P3
```



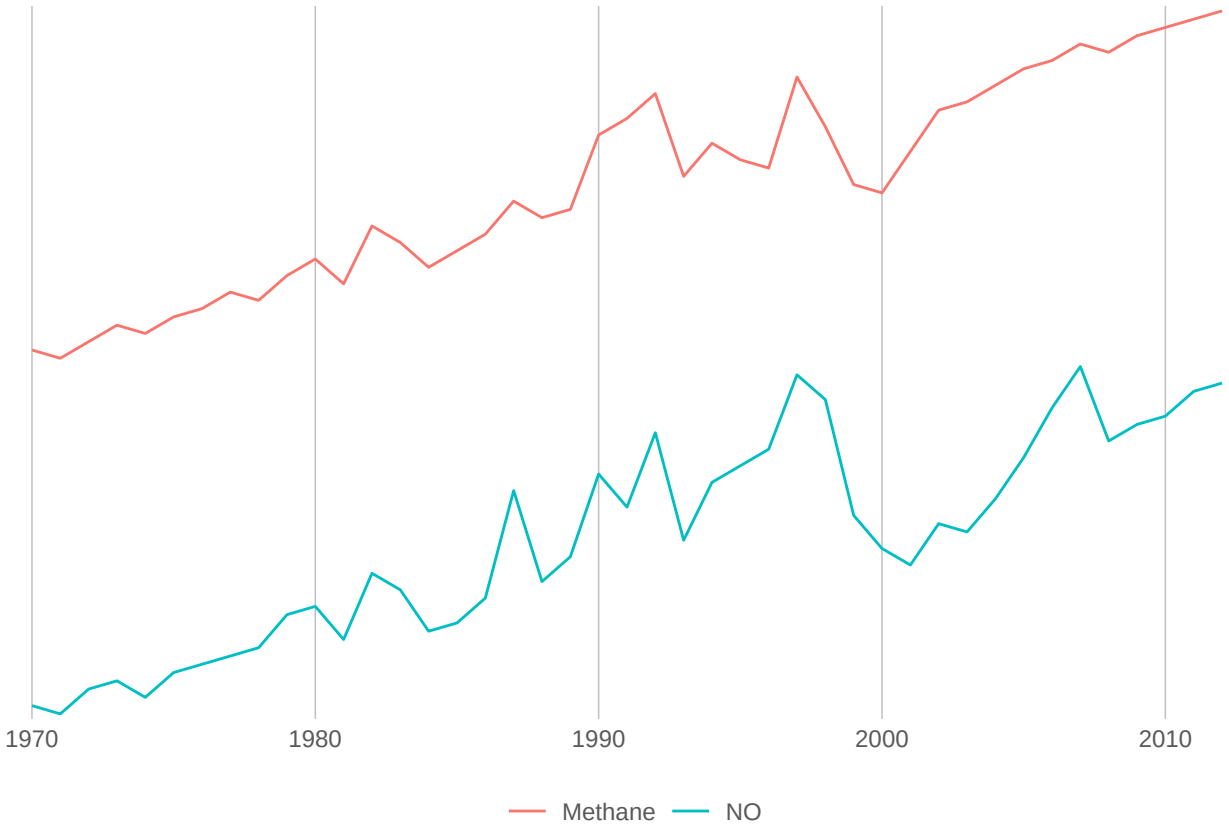
Cambio acumulado global del nivel del mar (1993-2015)

```
P2
```



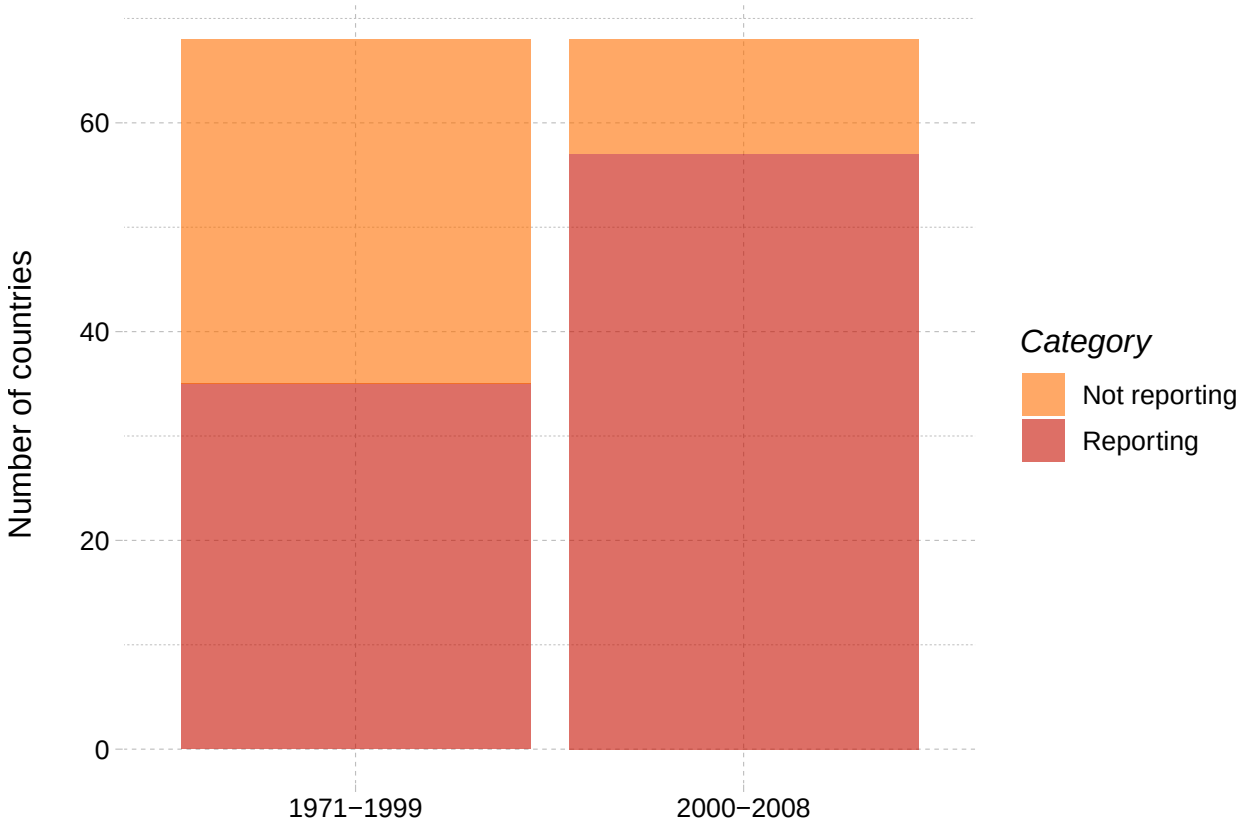
Emisiones de metano y NO (1979-2012)

```
P4
```



Países que notifican temperaturas extremas

P5


Number of countries
60
40
20
0
1971−1999
2000−2008
Category
Not reporting
Reporting