Informe_R_Final_

Franco Santilli[a,1,*], Guillermo Zaina[a], Marcos Santiago[a,1], Renzo Tagarot[a,1], Fernando Rodriguez[a,1]

[a] *Paseo Dr.~Emilio Descotte 750 M5500 Mendoza*

Abstract

Metodo sys.time

Esto lo utilizaremos para medir el tiempo de ejecución de un fragmento de código, colocándolo al principio y fin del fragmento.

```
duermete_un_minuto <- function() { Sys.sleep(5) }
start_time <- Sys.time()
duermete_un_minuto()
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 5.106807 secs
```

Con esto, hemos generaos una función de tiempo que antes no existía.

Metodo biblioteca tictoc

Esto de usar una biblioteca es llamar u cargar una procedimientos que generará comando nuevos en R.

```
library(tictoc)
tic("sleeping")
A<-20
print("dormire una siestita...")
```

```
## [1] "dormire una siestita..."
```

*Corresponding author

Email addresses: francosantilli47@gmail.com (Franco Santilli), zainaguillermo@gmail.com (Guillermo Zaina), marcossantiagoh2@gmail.com (Marcos Santiago), renzot0800@gmail.com (Renzo Tagarot), ferodriguez15@gmail.com (Fernando Rodriguez)

```
Sys.sleep(2)
print("...suena el despertador")
```

```
## [1] "...suena el despertador"
```

```
toc()
```

```
## sleeping: 2.11 sec elapsed
```

De igual manera, solo se podrá cronometrar un fragmento de código a la vez.

Metodo biblioteca rbenchmark

R lo llama como "un simple contenedor alrededor del fragmento de codigo system.time". Sin embargo agrega mas conveniencia a este, como por ejemplo para cronometrar múltiples expresiones se necesita únicamente una sola llamada de referencia. También los resultados se organizan en un marco de datos, entre otras.

```
library(rbenchmark)
# lm crea una regresion lineal
benchmark("lm" = {
X <- matrix(rnorm(1000), 100, 10)
y <- X %*% sample(1:10, 10) + rnorm(100)
b <- lm(y ~ X + 0)$coef
},
"pseudoinverse" = {
X <- matrix(rnorm(1000), 100, 10)
y <- X %*% sample(1:10, 10) + rnorm(100)
b <- solve(t(X) %*% X) %*% t(X) %*% y
},
"linear system" = {
X <- matrix(rnorm(1000), 100, 10)
y <- X %*% sample(1:10, 10) + rnorm(100)
b <- solve(t(X) %*% X, t(X) %*% y)
},
replications = 1000,
columns = c("test", "replications", "elapsed",
"relative", "user.self", "sys.self"))
```

```
##            test replications elapsed relative user.self sys.self
## 3 linear system         1000    0.73    1.000      0.44     0.04
## 1            lm         1000    5.41    7.411      3.36     0.22
## 2 pseudoinverse         1000    0.95    1.301      0.54     0.01
```

Finalmente se nos entregará el tiempo cronometrado en cada parte del código como se puede ver en el cuadro.

Metodo biblioteca microbenchmark

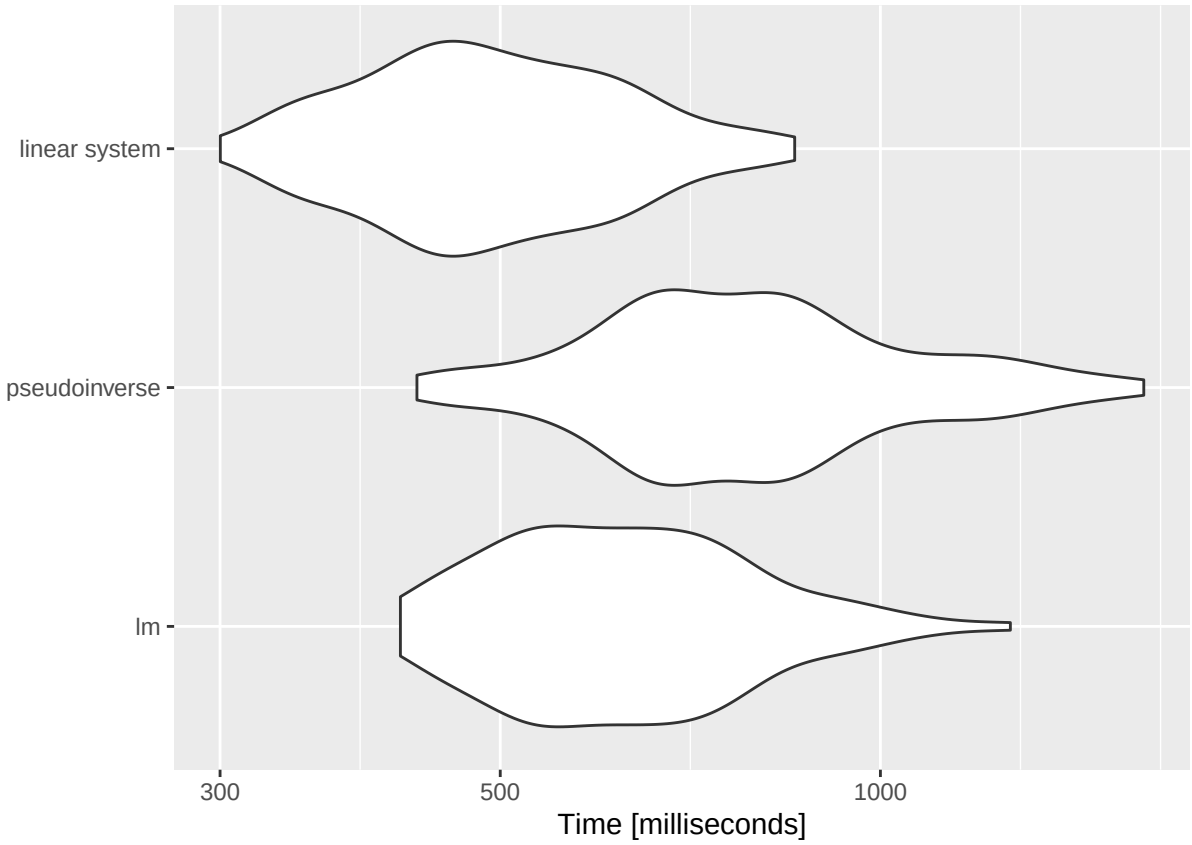
Es similar al paquete rbenchmark, ya que nos permite comparar tiempos cronometrados en múltiples fragmentos de código dentro de R. Sin embargo este presenta mayor comodidad y funcionalidad, aunque acabe siendo un poco inestable (aunque no sera un gran problema para el consumidor final). Algunas de estas nuevas funcionalidades son; por ejemplo, es que se puede ver a través de un cuadro la salida del código, como también el poder verificar automáticamente los resultados de las expresiones de referencia sin ser especificamente solicitadas.

```
library(microbenchmark)
set.seed(2017)
n <- 10000
p <- 100
X <- matrix(rnorm(n*p), n, p)
y <- X %*% rnorm(p) + rnorm(100)
check_for_equal_coefs <- function(values) {
tol <- 1e-12
max_error <- max(c(abs(values[[1]] - values[[2]]),
abs(values[[2]] - values[[3]]),
abs(values[[1]] - values[[3]])))
max_error < tol
}
mbm <- microbenchmark("lm" = { b <- lm(y ~ X + 0)$coef },
"pseudoinverse" = {
b <- solve(t(X) %*% X) %*% t(X) %*% y
},
"linear system" = {
b <- solve(t(X) %*% X, t(X) %*% y)
},
check = check_for_equal_coefs)
mbm
```

```
## Unit: milliseconds
##           expr      min       lq     mean   median       uq       max neval
##             lm 416.7854 523.5713 643.5524 619.4224 723.9560 1267.4070   100
##  pseudoinverse 429.5449 653.2967 818.6715 783.0602 898.7846 1616.3826   100
##  linear system 300.1373 422.9701 503.6173 481.5339 584.1457  855.3172   100
```

```
library(ggplot2)
autoplot(mbm)
```

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



Se puede ver, como se mencionó anteriormente, el tiempo cronometrado en cada fragmento de código, como también graficados los mismos posteriormente.

Trabajo de evaluacion del modulo

Comparar la generacion de un vector secuencia

Aunque en R exista la funcionalidad de generar una secuencia con el comando "seq", nosotros utilizaremos otras formas para generar vectores o secuencias.

Secuencia generada con for

A continuación generaremos un vector con el comando "for", agregando también "sys.time" para cronometrar el tiempo de generación del mismo.

```
start_time <- Sys.time()
for (i in 1:50000) { A[i] <- (i*2)}
head (A)
```

```
## [1]  2  4  6  8 10 12
```

```
tail(A)
```

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

```
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.174068 secs
```

He aquí los datos prometidos al principio.

Secuencia generada con R

A continuación generaremos un vector con el comando que mencionamos principalmente "seq", cronometrando también el tiempo de realización del mismo.

```
start_time <- Sys.time()
A <- seq(1,1000000, 2)
head (A)
```

```
## [1]  1  3  5  7  9 11
```

```
tail(A)
```

```
## [1] 999989 999991 999993 999995 999997 999999
```

```
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.06461 secs
```

A traves del comando "sys.time", podemos ver que para generar un vector secuencia (que en nuestro caso contiene numeros del 1 al 100.000 en intervalos de 2) se va a requerir un menor tiempo de procesamiento el comando para generar secuencias "seq" que ya viene de base con el RStudio. Aunque la diferencia entre "for" y "seq" es mínima, se denota cuál es más eficiente.

Definicion matematica recurrente

Ahora mostraremos como programar la serie de Fibonacci a través de una definición matemática recurrente. Esta serie de números es tal que comienza con 0 y 1, y a partir de estos, cada número es la suma de los 2 anteriores. Como dato a destacar, esta serie tiene numerosas aplicaciones en distintas ciencias de la computación, matemática y teoría de juegos.

```
start_time <- Sys.time()
for(i in 0:5)
{ a<-i
b <-i+1
c <- a+b
print(c)
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
## [1] 11
```

```
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.1195741 secs
```

```
start_time <- Sys.time()
f1<-0
f2<-1
N<-0
vec<- c(f1,f2)
f3<-0

while (f3 <= 1000000) {
  N<-N+1
  f3<-f1++f2
  vec<- c(vec,f3)
  f1<-f2
  f2<-f3
  c<-a+b
  i<-i+1
}
N
```

```
## [1] 30
```

```
vec
```

```
##  [1]       0       1       1       2       3       5       8      13      21
## [10]      34      55      89     144     233     377     610     987    1597
## [19]    2584    4181    6765   10946   17711   28657   46368   75025  121393
## [28]  196418  317811  514229  832040 1346269
```

```
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.185205 secs
```

Finalmente para generar un número mayor a 1.000.000 en la serie, se necesitan 30 iteraciones.

Ordenación de un vector por metodo burbuja

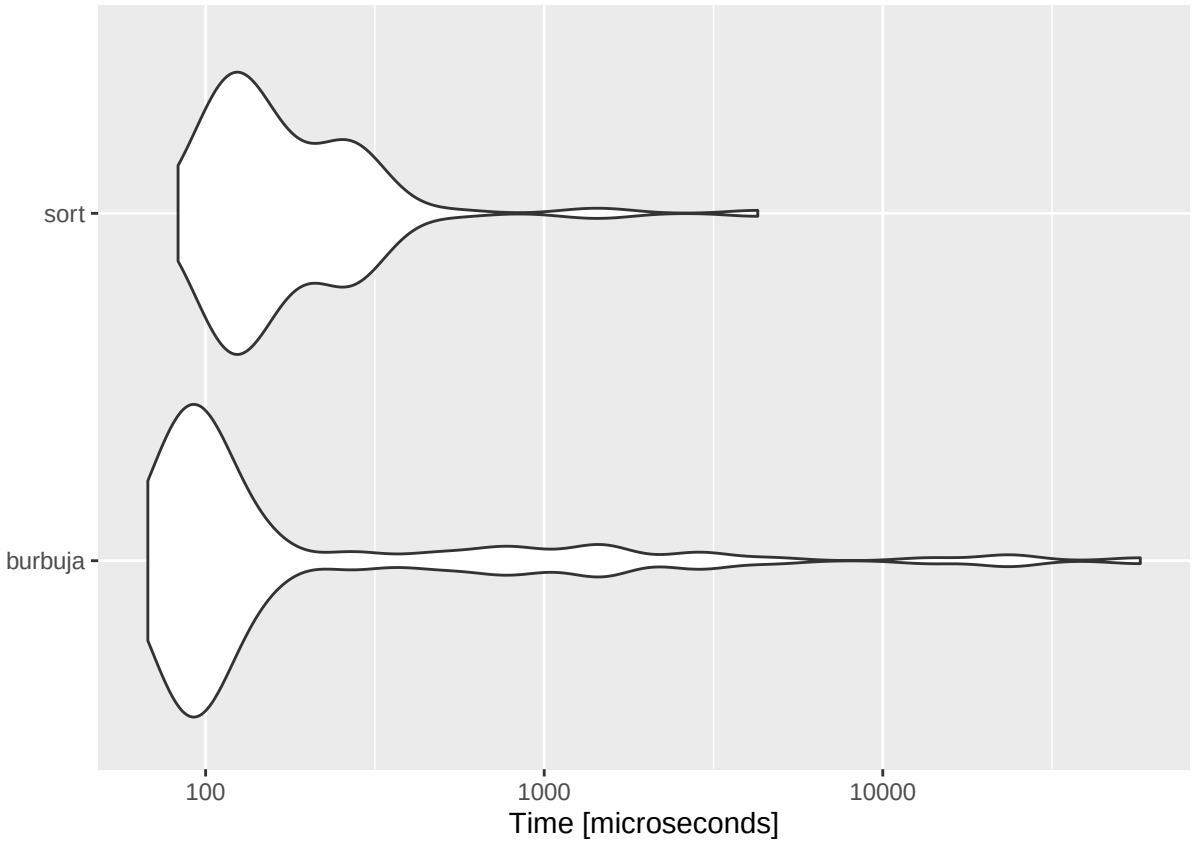
Este es un simple método de ordenación, también conocido como método del intercambio directo. Funciona de manera que revisa cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

```
library(microbenchmark)

x<-sample(1:100,20)


mbm <- microbenchmark(
  "burbuja"={

burbuja <- function(x){
n<-length(x)
for(j in 1:(n-1)){
for(i in 1:(n-j)){
if(x[i]>x[i+1]){
temp<-x[i]
x[i]<-x[i+1]
x[i+1]<-temp
}
}
}
return(x)
}
res<-burbuja(x)

  },

"sort" = {
  sort(x)
}
)

mbm
```

```
## Unit: microseconds
##     expr  min     lq     mean median     uq     max neval
##  burbuja 67.5  86.15 1541.536 102.35 211.90 57635.2   100
##     sort 82.9 117.40  241.882 141.35 249.15  4274.1   100
```

```
library(ggplot2)
autoplot(mbm)
```

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



Comparando los dos métodos a través de microbenchmark podemos notar que el metodo de la burbuja requiere de mucho más recursos y tiempo de procesamiento que el metodo sort. Sin embargo se puede observar una mayor presición en el método de la burbuja.

Modelado matemático de una epidema

```
# Numeros de casos semanales en Argentina
f1<-51778

N<-0
f3 <- 1.62
f2 <- 0
vec <- c(f1)

while (f1 <= 40000000) {
  f2 <- f1*f3
  f1 <- f2
  N <- N+1
  vec <- c(vec,f2)

}

N
```

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

```
vec
```

```
##  [1]    51778.00    83880.36   135886.18   220135.62   356619.70   577723.91
##  [7]   935912.74  1516178.64  2456209.39  3979059.21  6446075.93 10442643.00
## [13] 16917081.66 27405672.29 44397189.11
```

Con el número de casos actuales semanales en Argentina y con el factor de contagio alrededor de F=1.62, tardariamos 14 semanas en llegar a los 40 millones de contagiados .

Importar datos de la red o de excel

Cabe mencionar que para poder importar un archivo de excel (u otro programa), primeramente se necesita realizar una serie de pasos. Estos son: 1. Ir a "File" 2. Ir a "Import Dataset" 3. Click en "From Text (readr)" 4. A continuación en la pestaña seleccionar el archivo de excel buscado a traves de "browse". Se cargará el archivo. 5. Finalmente en "Delimiter" seleccionar "Semicolon" 6. Click en "Import", lo que generará el archivo en una pestaña de R. 7. Así luego escribimos el siguiente fragmento de código para añadirlo a este código:

```
library(readr)
casos <- read_delim("C:/Users/franc/Downloads/casos.csv",
    delim = ";", escape_double = FALSE, col_types = cols(`Covid Argentina` = col_date(format
    trim_ws = TRUE)
```

```
## New names:
## * `` -> `...2`
## * `` -> `...3`
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

Grafico de barras en caso covid

```
library(readr)
casos <- read_delim("C:/Users/franc/Downloads/casos.csv",
    delim = ";", escape_double = FALSE, col_types = cols(...2 = col_number()),
    trim_ws = TRUE)
```

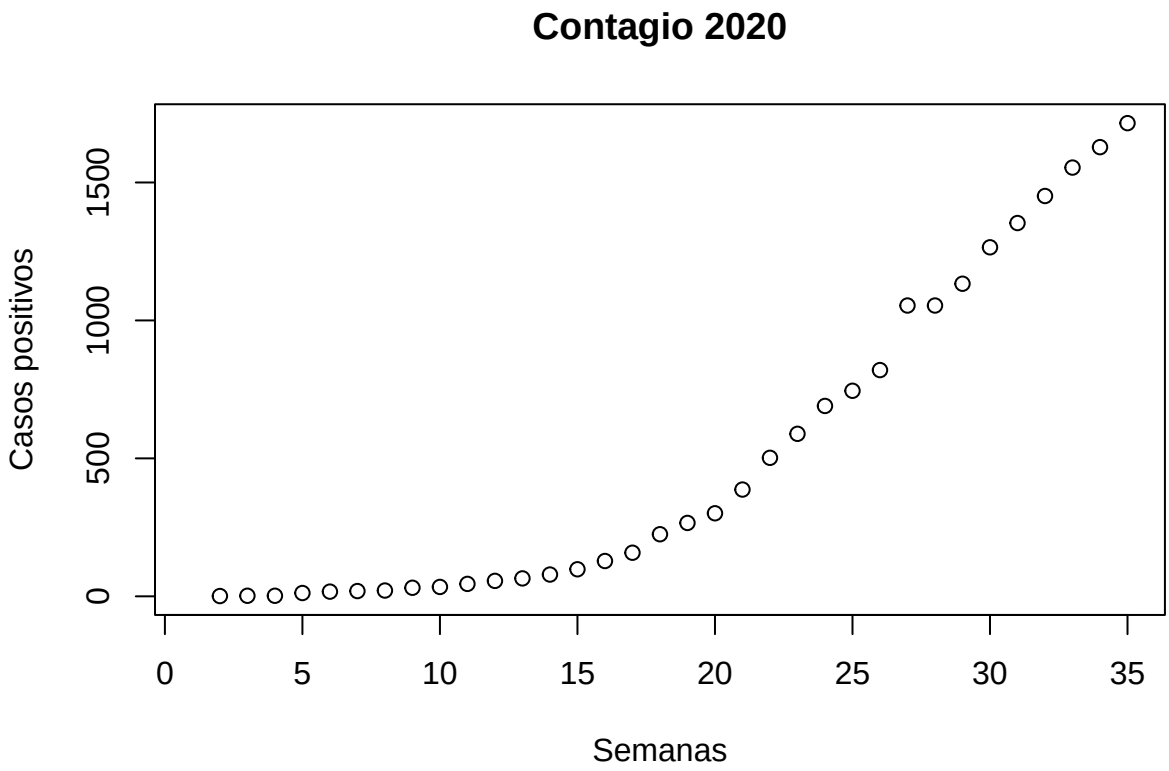
```
## New names:
## * `` -> `...2`
## * `` -> `...3`
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

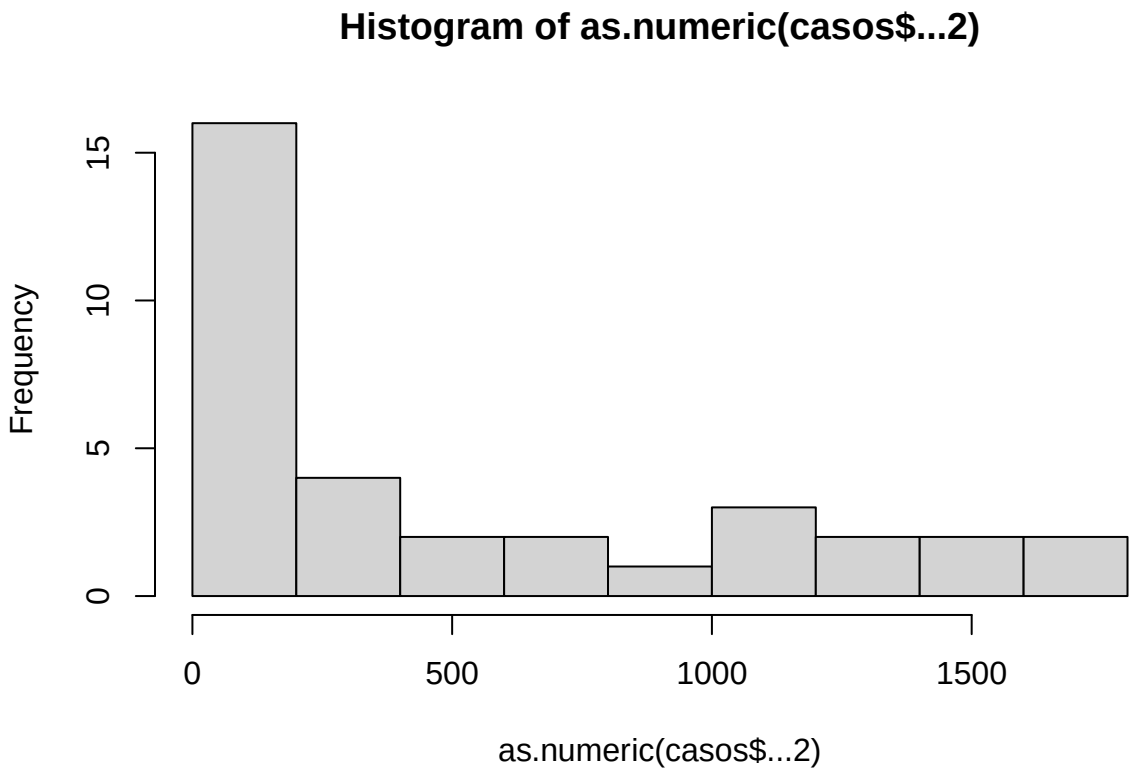
```
casos$...2
```

```
##  [1]   NA    1    2    2   12   17   19   21   31   34   45   56   65   79   98
## [16]  128  158  225  266  301  387  502  589  690  745  820 1054 1054 1133 1265
## [31] 1353 1451 1554 1628 1715
```

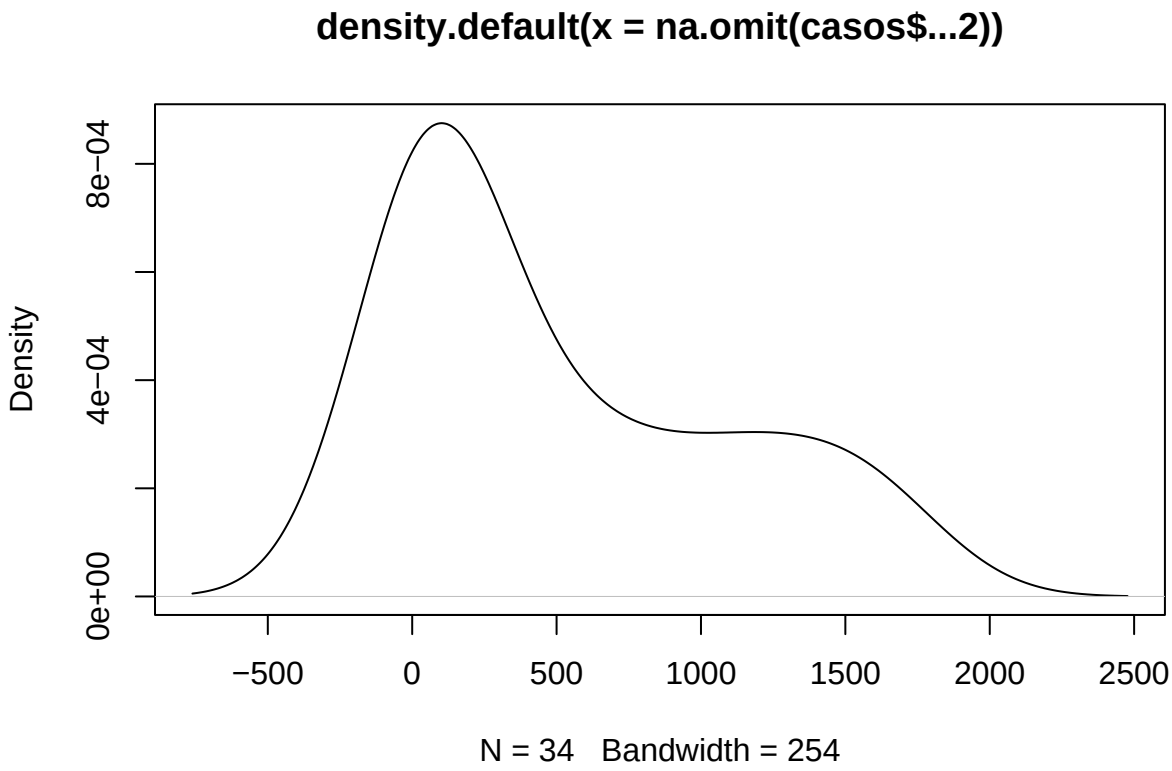
```
plot(casos$...2, main="Contagio 2020", ylab="Casos positivos", xlab="Semanas")
```




```
hist(as.numeric(casos$...2))
```




```
plot(density(na.omit(casos$...2)))
```



Finalmente gracias a la planilla de excel podemos observar el gráfico finalizado sobre los casos de covid.