Time Series Analysis: Descriptive Analysis

FSC

March 7, 2019

Contents

Que es una serie temporal?

Hablamos de una serie temporal específicamente si se trata de monitorizar algún tipo de variable a intervalos constantes en el tiempo, i.e. cad mes, cada cuatrimestre, cada semestre, cada año... Para mediciones irregulares utilizaremos otro tipo de análisis (medidas repetidas) que veremos mañana

Ejemplo motivacional

Desde la página *https* : *//data.worldbank.org/* podemos acceder a un gran número de estadísticas relacionadas con indicadores socioeconómicos que comparan a los países del mundo. Hans Rosling de la fundación gapminder fue uno de los inductores de esta iniciativa para desmitificar algunos de los mitos actuales acerca del progreso y las diferencias socioeconómicas entre países. Os recomiento la charla ted: *https: //www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen*

Esta página permite descargar muchos datos de su base de datos, usando el paquete "WDI":

```
# Loading data from the WDB web using an API
#install.packages('WDI')
#library(WDI)
```

```
#WDIsearch(string = "life.*expectancy", field = "name", cache = NULL)
#df.le = WDI(country = "all", indicator = c("SP.DYN.LE00.IN"), start = 1900,
#             end = 2012)
```

Alternativamente vamos a leerlo del fichero "LifeExpectancy.txt" que os he proporcionado

```
setwd("C:/Users/fscabo/Desktop/MasterDataScience_KSchool/Session7_TimeSeries_I")
df.le=read.delim("code_TS/LifeExpectancy.txt",header=T,sep="\t")
head(df.le)
```

```
##   iso2c    country SP.DYN.LE00.IN year
## 1    1A Arab World       70.40802 2012
## 2    1A Arab World       70.22389 2011
## 3    1A Arab World       70.04105 2010
## 4    1A Arab World       69.85143 2009
## 5    1A Arab World       69.64697 2008
## 6    1A Arab World       69.42455 2007
```

```
is.data.frame(df.le)
```

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

```
head(levels(factor(df.le$country)))
```

```
## [1] "Afghanistan"    "Albania"        "Algeria"        "American Samoa"
## [5] "Andorra"        "Angola"
```

```
levels(factor(df.le$year))
```

```
##  [1] "1960" "1961" "1962" "1963" "1964" "1965" "1966" "1967" "1968" "1969"
## [11] "1970" "1971" "1972" "1973" "1974" "1975" "1976" "1977" "1978" "1979"
## [21] "1980" "1981" "1982" "1983" "1984" "1985" "1986" "1987" "1988" "1989"
## [31] "1990" "1991" "1992" "1993" "1994" "1995" "1996" "1997" "1998" "1999"
## [41] "2000" "2001" "2002" "2003" "2004" "2005" "2006" "2007" "2008" "2009"
## [51] "2010" "2011" "2012"
```

En una primera aproximación usamos los años simplemente como una variable categórica y dibujamos los boxplots de las esperanzas de vida:

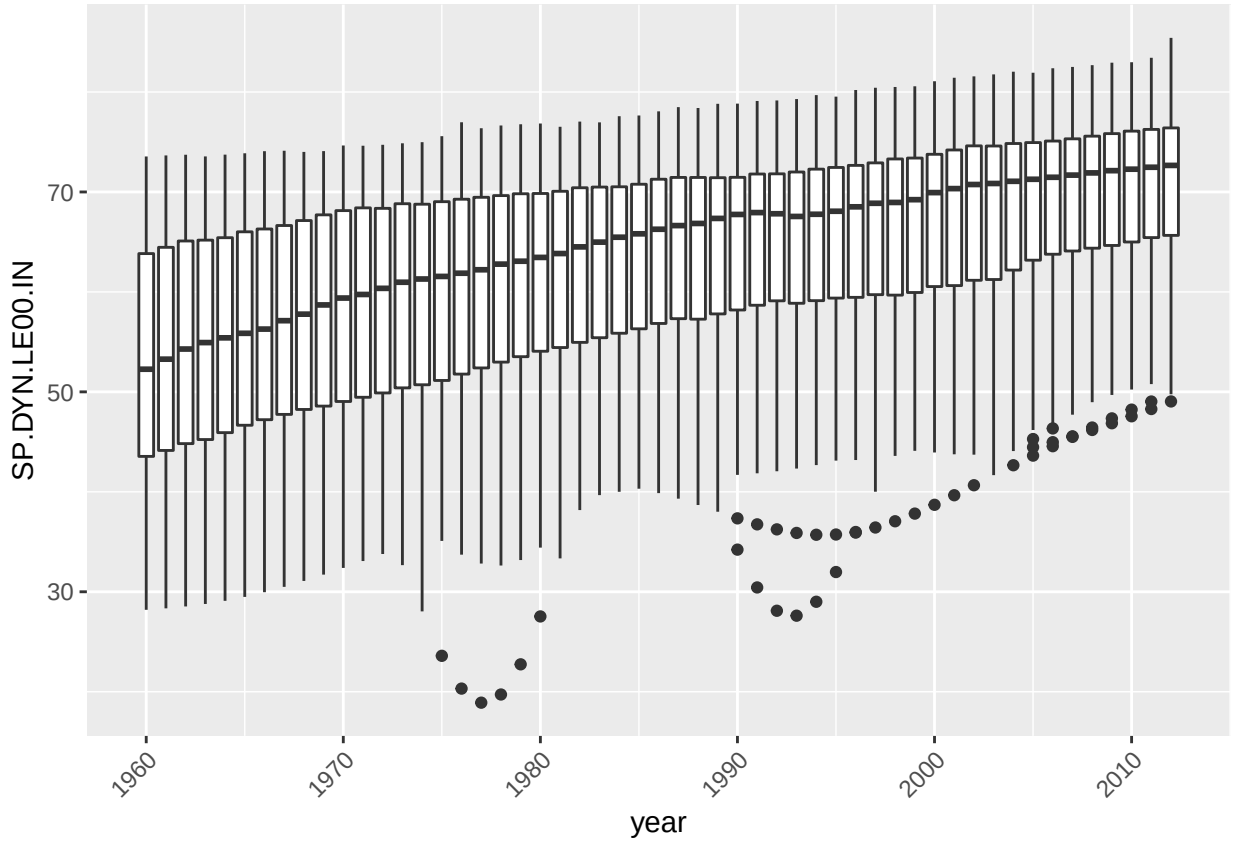
```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

```
g = ggplot() + geom_boxplot(data = df.le, aes(x = year, y = SP.DYN.LE00.IN,
                                              group = year))
g = g + theme(axis.text.x = element_text(angle = 45, hjust = 1))
g
```

```
## Warning: Removed 1224 rows containing non-finite values (stat_boxplot).
```



Vamos a intentar identificar esos paises "outlier" en los que alrededor de 1988 la esperanza de vida era menor de 40 años:

```
subset(df.le, year > 1988 & SP.DYN.LE00.IN < 40)

g = g + geom_line(data = subset(df.le, country == "Rwanda"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "red")
g

g = g + geom_line(data = subset(df.le, country == "Sierra Leone"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "orange")
g

# H01: Genocide > 1994 x
# H02: AIDS epidemy also in Kenya, South Africa, Uganda, etc
g = g + geom_line(data = subset(df.le, country =="Kenya"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "green")
g

g = g + geom_line(data = subset(df.le, country =="South Africa"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "green")
g

g = g + geom_line(data = subset(df.le, country =="Uganda"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "green")
g
```

```
#H03: Civil War
g = g + geom_line(data = subset(df.le, country =="Bangladesh"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "blue")
g

g = g + geom_line(data = subset(df.le, country =="Iraq"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "red")
g

g = g + geom_line(data = subset(df.le, country =="Iran, Islamic Rep."), aes(x = year, y = SP.DYN.LE00.I
                  col = "red",lty=2)
g

g = g + geom_line(data = subset(df.le, country =="Afghanistan"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "violet")
g

g = g + geom_line(data = subset(df.le, country =="Cambodia"), aes(x = year, y = SP.DYN.LE00.IN),
                  col = "pink")
g
```

Fechas, datos y series temporales en R:

Leyendo fechas en R: el tipo de dato "date"

En R tenemos fundamentalmente tres tipos de vectores: numérico, carácter y lógico. Acabamos de leer datos de años usándolos como si fueran strings. Sin embargo, esto no permite hacer operaciones con las fechas, limitando nuestros análisis.

R define específicamente un tipo para fechas e incluso horas. Veamos un ejemplo con los resultados electorales de EEUU en 2016 ($https://rafalab.github.io/dsbook/parsing-dates-and-times.html$)

```
library(tidyverse)
library(dslabs)
data("polls_us_election_2016")
polls_us_election_2016$startdate %>% head
```

```
## [1] "2016-11-03" "2016-11-01" "2016-11-02" "2016-11-04" "2016-11-03"
## [6] "2016-11-03"
```

Aunque parezcan strings no lo son:

```
class(polls_us_election_2016$startdate)
```

```
## [1] "Date"
```

Qué sucede si queremos transformarlos en números:

```
as.numeric(polls_us_election_2016$startdate) %>% head
```

```
## [1] 17108 17106 17107 17109 17108 17108
```

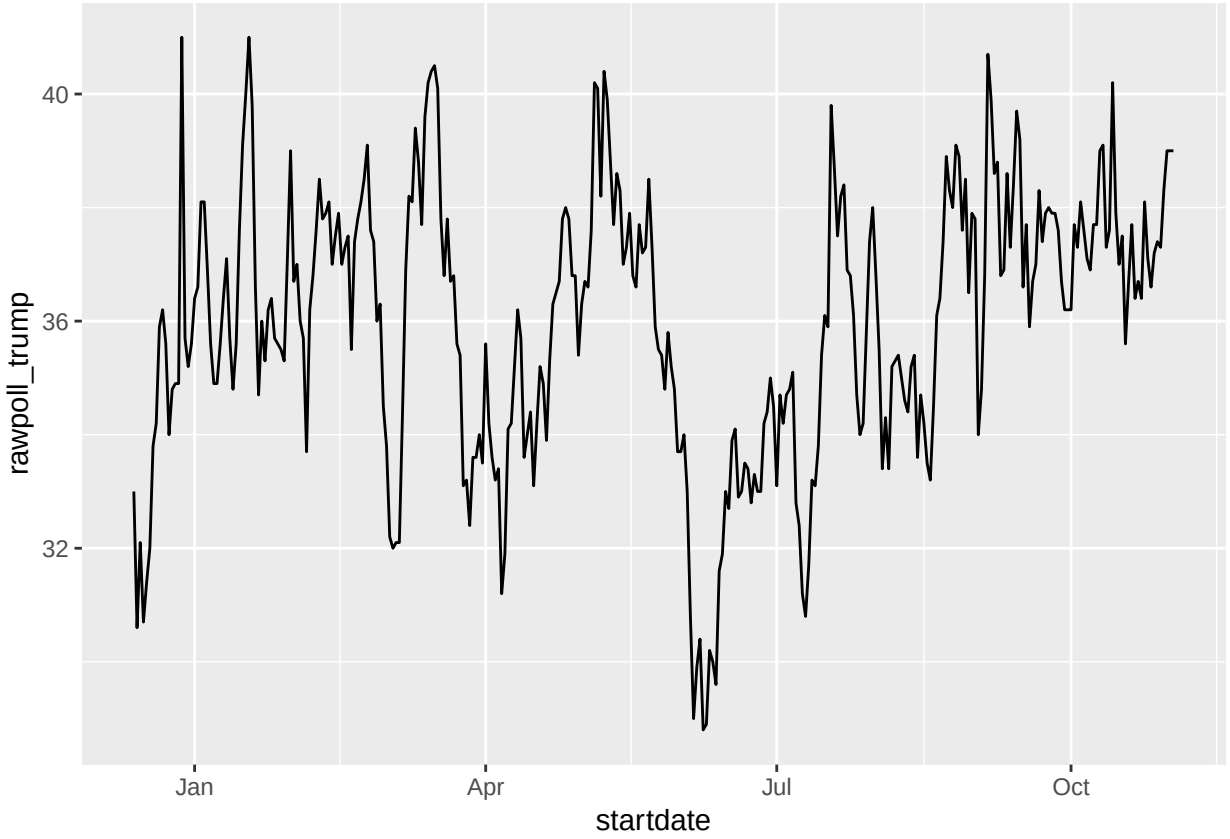
Son los días transcurridos desde el día 0 o *epoch* que suele ser el 1/01/1970 en la mayor parte de los lenguajes de programacion.

```
as.Date("1970-01-01") %>% as.numeric
```

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

ggplot identifica este formato como fechas. Por ejemplo en un scatterplot:

```
polls_us_election_2016 %>% filter(pollster == "Ipsos" & state =="U.S.") %>%
  ggplot(aes(startdate, rawpoll_trump)) +
  geom_line()
```



Automáticamente muestra los meses, lo que es muy útil.

The lubridate package

El entorno de funciones tidyverse incluye numerosas funciones para trabajar con fechas a través del paquete **lubridate**.

```
library(lubridate)
```

Podemos tomar una muestra aleatoria de fechas y ordenarlas:

```
set.seed(2002)
dates <- sample(polls_us_election_2016$startdate, 10) %>% sort
dates
```

```
##  [1] "2015-11-10" "2015-12-04" "2016-04-11" "2016-05-16" "2016-08-17"
##  [6] "2016-08-19" "2016-09-01" "2016-10-21" "2016-10-25" "2016-11-01"
```

`year`, `month` y `day` son funciones para extraer dicha información de fechas:

```
data_frame(date = dates,
           month = month(dates),
           day = day(dates),
           year = year(dates))
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
## # A tibble: 10 x 4
##    date       month   day  year
##    <date>     <dbl> <int> <dbl>
##  1 2015-11-10    11    10  2015
##  2 2015-12-04    12     4  2015
##  3 2016-04-11     4    11  2016
##  4 2016-05-16     5    16  2016
##  5 2016-08-17     8    17  2016
##  6 2016-08-19     8    19  2016
##  7 2016-09-01     9     1  2016
##  8 2016-10-21    10    21  2016
##  9 2016-10-25    10    25  2016
## 10 2016-11-01    11     1  2016
```

Podemos extraer las etiquetas de los meses:

```
month(dates, label = TRUE)
```

```
##  [1] Nov Dec Apr May Aug Aug Sep Oct Oct Nov
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

Otra función interesantes son los *parsers* que convierten *strings* en *dates*. La función `ymd` asume que las fechas están en formato $YYYY - MM - DD$ e intentan parseae a formato fecha de la mejor forma posible:

```
x <- c(20090101, "2009-01-02", "2009 01 03", "2009-1-4",
       "2009-1, 5", "Created on 2009 1 6", "200901 !!! 07")
ymd(x)
```

```
## [1] "2009-01-01" "2009-01-02" "2009-01-03" "2009-01-04" "2009-01-05"
## [6] "2009-01-06" "2009-01-07"
```

Sea cual sea el formato de tu fecha **lubridate** tiene un parser para convertirlo en dato fecha.

For example, if the string is:

```
x <- "09/01/02"
```

La función `ymd` asume que el primer elemento de la entrada es el año, luego el mes y finalmente el día y lo convierte a:

```
ymd(x)
```

```
## [1] "2009-01-02"
```

`mdy` asume que la primera entrada es el mes, luego el dia y al final el año:

```
mdy(x)
```

```
## [1] "2002-09-01"
```

Más posibilidades:

```
ydm(x)
```

```
## [1] "2009-02-01"
```

```
myd(x)
```

```
## [1] "2001-09-02"
```

```
dmy(x)
```

```
## [1] "2002-01-09"
```

```
dym(x)
```

```
## [1] "2001-02-09"
```

A parte de poder trabajar tambien con horas y franjas horarias el paquete tiene una función `make_date` para crear un objeto *date*. Toma tres argumentos: año, mes, dia, hora, minutos y segundos. March 8, 2019 escribiríamos:

```
make_date(2019, 8, 3)
```

```
## [1] "2019-08-03"
```

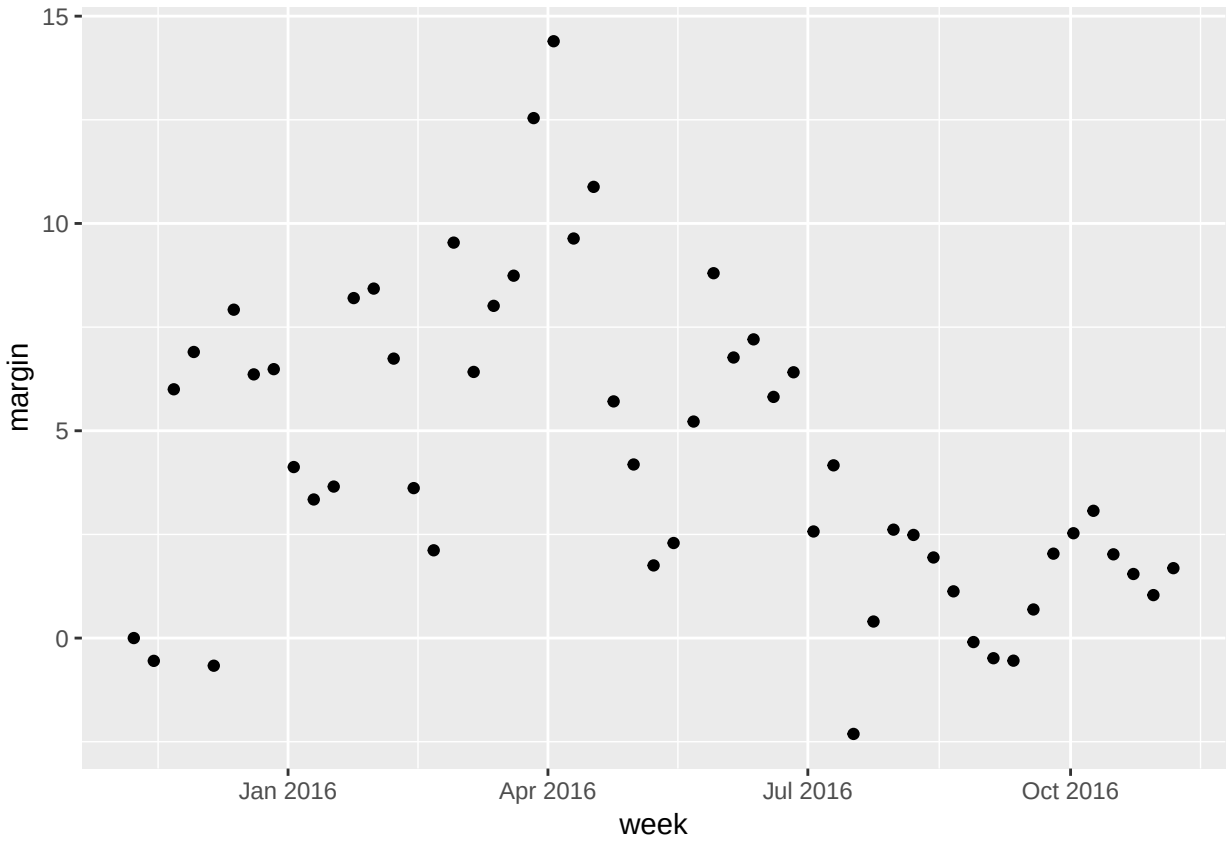
Para hacer un vector del 1 de enero de los 80s:

```
make_date(1980:1989)
```

```
##  [1] "1980-01-01" "1981-01-01" "1982-01-01" "1983-01-01" "1984-01-01"
##  [6] "1985-01-01" "1986-01-01" "1987-01-01" "1988-01-01" "1989-01-01"
```

`round_date` redondea fechas al próximo año, mes, dia... En el ejemplo de las elecciones si queremo redondear las votaciones por semana del año podemos hacer lo siguiente:

```
polls_us_election_2016 %>%
  mutate(week = round_date(startdate, "week")) %>%
  group_by(week) %>%
  summarize(margin = mean(rawpoll_clinton - rawpoll_trump)) %>%
  qplot(week, margin, data = .)
```



Ejercicio

De un pdf hemos extraido estadísticas de puerto rico en los últimos 12 meses.

```
library(tidyverse)
library(purrr)
library(pdftools)
```

```
## Warning: package 'pdftools' was built under R version 3.5.2
```

```
fn <- system.file("extdata", "RD-Mortality-Report_2015-18-180531.pdf", package="dslabs")
tab <- map_df(str_split(pdf_text(fn), "\n"), function(s){
  s <- str_trim(s)
  header_index <- str_which(s, "2015")[1]
  tmp <- str_split(s[header_index], "\\s+", simplify = TRUE)
  month <- tmp[1]
  header <- tmp[-1]
  tail_index  <- str_which(s, "Total")
  n <- str_count(s, "\\d+")
  out <- c(1:header_index, which(n==1), which(n>=28), tail_index:length(s))
  s[-out] %>%
    str_remove_all("[^\\d\\s]") %>%
    str_trim() %>%
    str_split_fixed("\\s+", n = 6) %>%
    .[,1:5] %>%
    as_data_frame() %>%
    setNames(c("day", header)) %>%
```

```
    mutate(month = month,
           day = as.numeric(day)) %>%
    gather(year, deaths, -c(day, month)) %>%
    mutate(deaths = as.numeric(deaths))
})
```

```
## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

1. Haz un gráfico del número de muertes por fecha. Hint: convierte la variable meses de caracteres a números usando `recode` para redefinir la variable `tab`.

```
levels(as.factor(tab$month))
```

```
##  [1] "AGO" "APR" "DEC" "FEB" "JAN" "JUL" "JUN" "MAR" "MAY" "NOV" "OCT"
## [12] "SEP"
```

```
month.numeric=recode(tab$month,JAN="1",FEB="2",MAR="3",APR="4",MAY="5",JUN="6",
                               JUL="7",AGO="8",SEP="9",OCT="10",NOV="11",DIC="12")
tab<-mutate(tab,month_numeric=month.numeric)
```

2. Crea una nueva columna "date" con la fecha de cada entrada. Hint: use the `make_date` function.

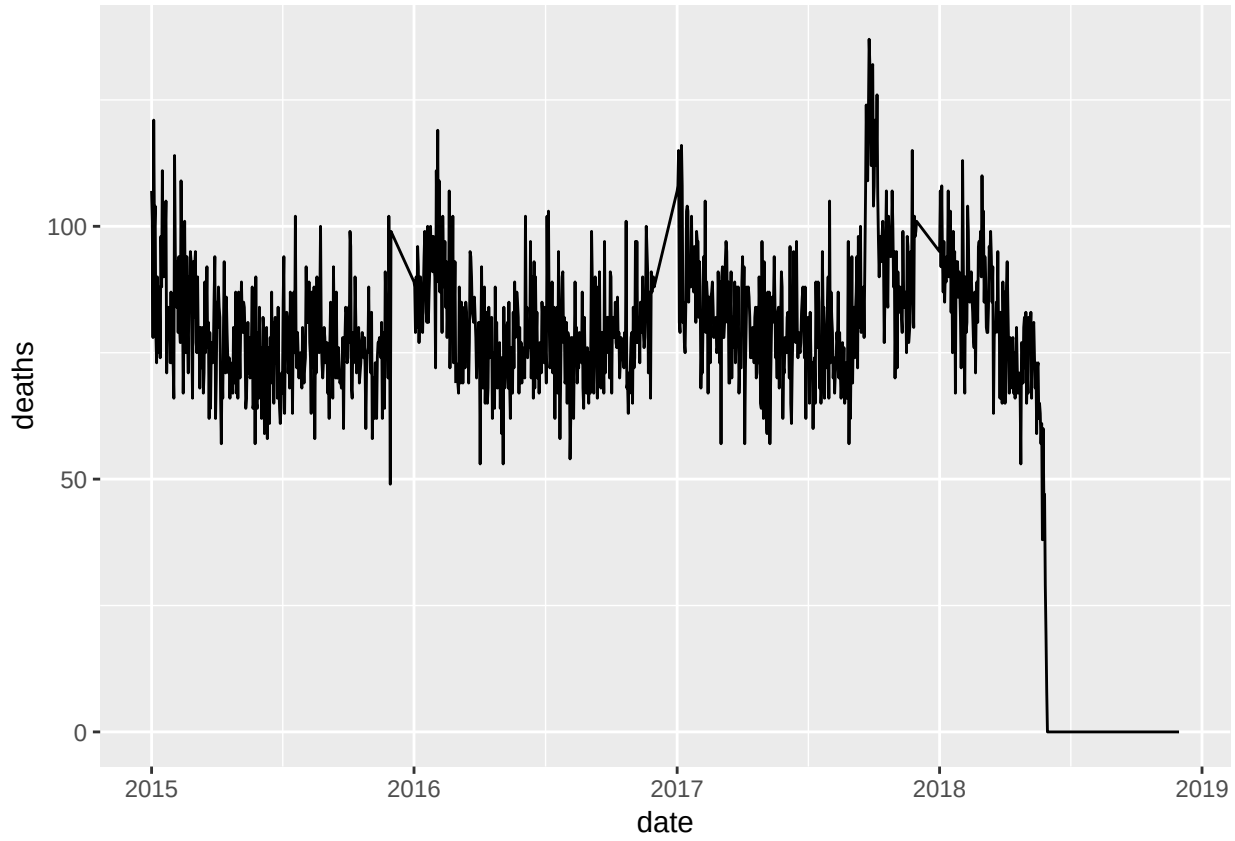
```
tab<-mutate(tab,date=make_date(year,month_numeric,day))
```

```
## Warning in make_date(year, month_numeric, day): NAs introduced by coercion
```

3. Deaths vs date.

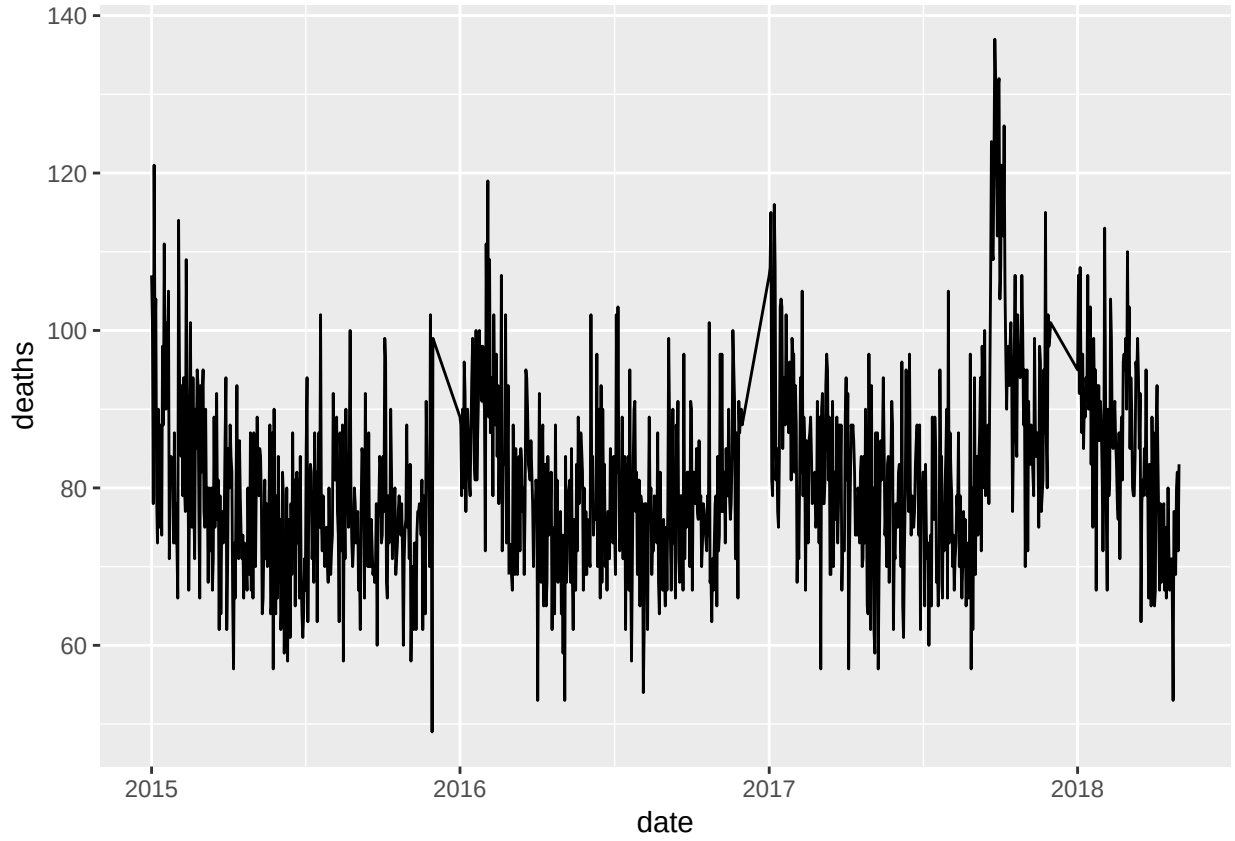
```
tab %>% ggplot(aes(date, deaths)) + geom_line()
```

```
## Warning: Removed 123 rows containing missing values (geom_path).
```



4. Probablemente las observaciones después de mayo de 2018 no se tomaron. Rehaz el plot sin esas observaciones

```
tab %>% filter(date <= "2018-05-01") %>% ggplot(aes(date, deaths)) + geom_line()
```



Pintando y manejando series temporales en R

Ahora que sabemos manejar datos de fechas vamos a ver qué tipo de dato podemos usar para almacenar toda una serie temporal y poder analizarla de manera sencilla. Vamos a utilizar tres datasets del paquete *fpp*2 desarrollado por Rob Jhyndman para aprender estos conceptos *https* : *//robjhyndman.com/teaching/*. El primero describe la edad de muerte de los reyes de Inglaterra. Leemos los datos de un archivo en la red y creamos un objeto *ts*() con el que podamos trabajar. Podemos representarlo mediante la función ts.plot()

```
library("fpp2")
```

```
## Warning: package 'fpp2' was built under R version 3.5.2
```

```
## Loading required package: forecast
```

```
## Warning: package 'forecast' was built under R version 3.5.2
```

```
## Loading required package: fma
```

```
## Warning: package 'fma' was built under R version 3.5.2
```

```
## Loading required package: expsmooth
```

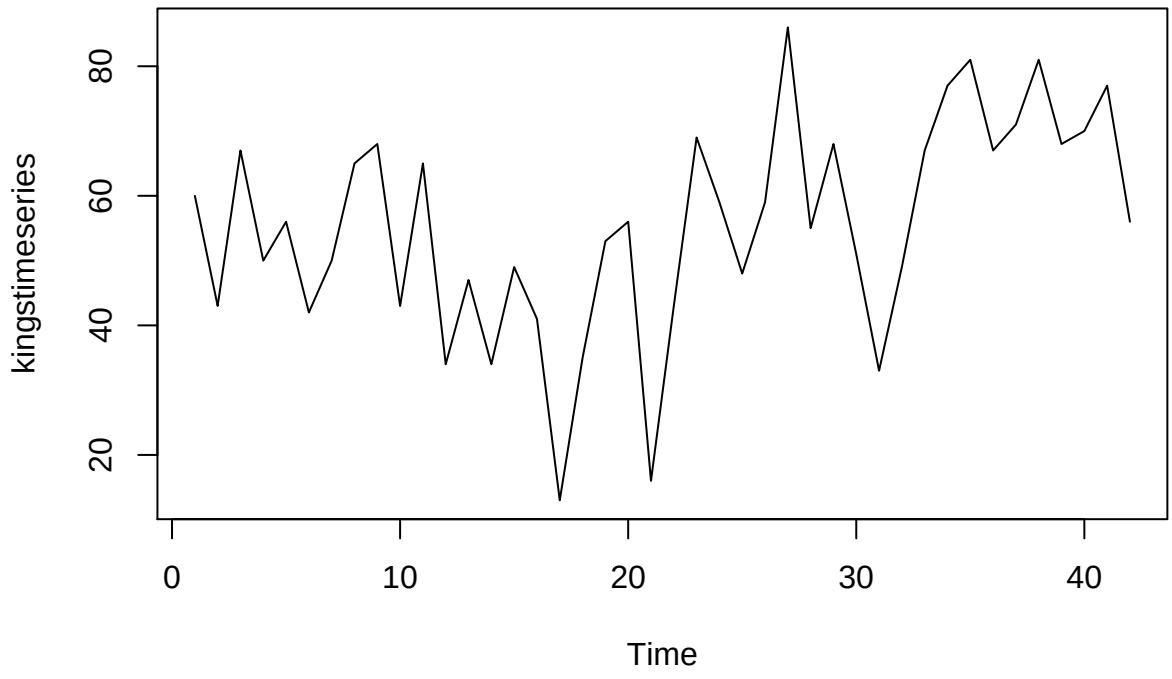
```
## Warning: package 'expsmooth' was built under R version 3.5.2
```

```
kings <- scan("http://robjhyndman.com/tsdldata/misc/kings.dat",skip=3)
kingstimeseries <- ts(kings)
kingstimeseries
```

```
## Time Series:
## Start = 1
```

```
## End = 42
## Frequency = 1
##  [1] 60 43 67 50 56 42 50 65 68 43 65 34 47 34 49 41 13 35 53 56 16 43 69
## [24] 59 48 59 86 55 68 51 33 49 67 77 81 67 71 81 68 70 77 56
```

```
ts.plot(kingstimeseries)
```



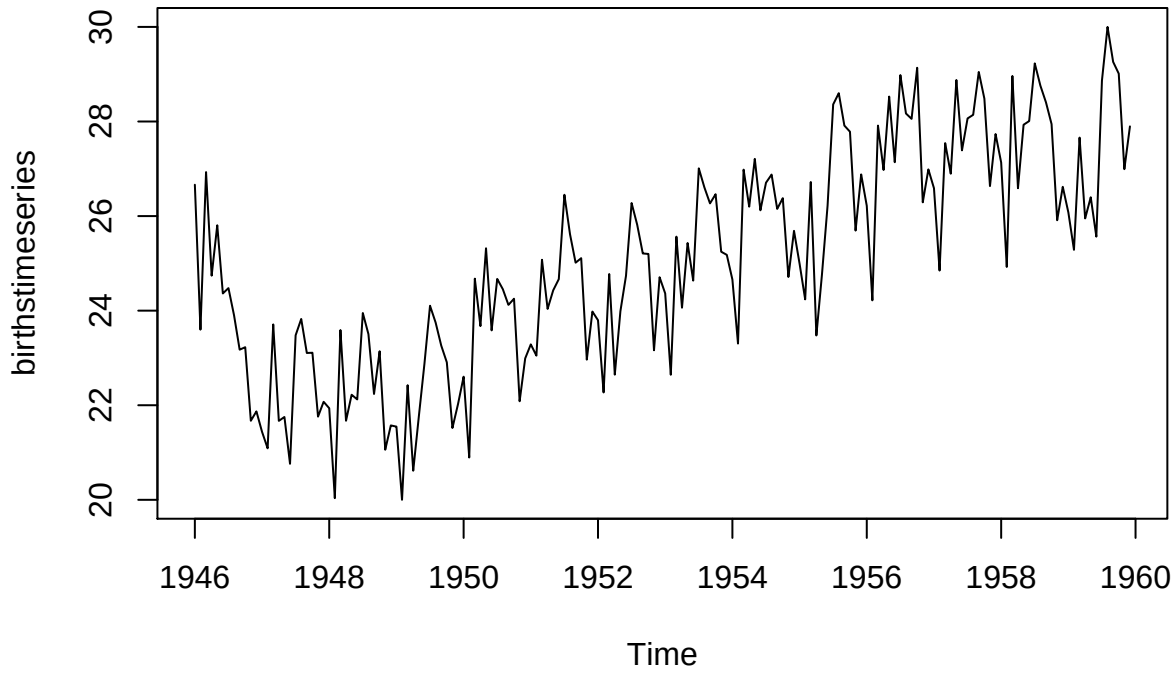
Hacemos lo mismo para un dataset que contiene información acerca de los nacimientos en NY:

```
births <- scan("http://robjhyndman.com/tsdldata/data/nybirths.dat")
birthstimeseries <- ts(births, frequency=12, start=c(1946,1))
birthstimeseries
```

```
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 1946 26.663 23.598 26.931 24.740 25.806 24.364 24.477 23.901 23.175 23.227
## 1947 21.439 21.089 23.709 21.669 21.752 20.761 23.479 23.824 23.105 23.110
## 1948 21.937 20.035 23.590 21.672 22.222 22.123 23.950 23.504 22.238 23.142
## 1949 21.548 20.000 22.424 20.615 21.761 22.874 24.104 23.748 23.262 22.907
## 1950 22.604 20.894 24.677 23.673 25.320 23.583 24.671 24.454 24.122 24.252
## 1951 23.287 23.049 25.076 24.037 24.430 24.667 26.451 25.618 25.014 25.110
## 1952 23.798 22.270 24.775 22.646 23.988 24.737 26.276 25.816 25.210 25.199
## 1953 24.364 22.644 25.565 24.062 25.431 24.635 27.009 26.606 26.268 26.462
## 1954 24.657 23.304 26.982 26.199 27.210 26.122 26.706 26.878 26.152 26.379
## 1955 24.990 24.239 26.721 23.475 24.767 26.219 28.361 28.599 27.914 27.784
## 1956 26.217 24.218 27.914 26.975 28.527 27.139 28.982 28.169 28.056 29.136
## 1957 26.589 24.848 27.543 26.896 28.878 27.390 28.065 28.141 29.048 28.484
## 1958 27.132 24.924 28.963 26.589 27.931 28.009 29.229 28.759 28.405 27.945
## 1959 26.076 25.286 27.660 25.951 26.398 25.565 28.865 30.000 29.261 29.012
```

```
##         Nov    Dec
## 1946 21.672 21.870
## 1947 21.759 22.073
## 1948 21.059 21.573
## 1949 21.519 22.025
## 1950 22.084 22.991
## 1951 22.964 23.981
## 1952 23.162 24.707
## 1953 25.246 25.180
## 1954 24.712 25.688
## 1955 25.693 26.881
## 1956 26.291 26.987
## 1957 26.634 27.735
## 1958 25.912 26.619
## 1959 26.992 27.897
```

```
ts.plot(birthstimeseries)
```



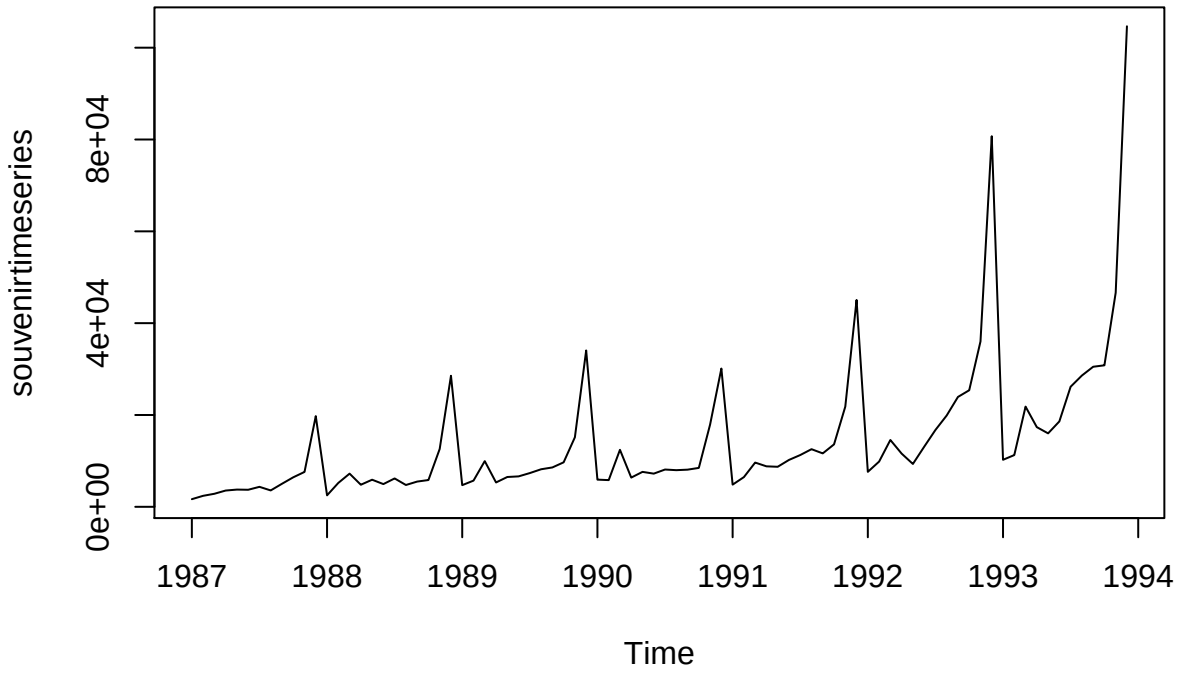
Y un último con las ventas mensuales de una tienda de souvenirs en Australia

```
souvenir <- scan("http://robjhyndman.com/tsdldata/data/fancy.dat")
souvenirtimeseries <- ts(souvenir, frequency=12, start=c(1987,1))
souvenirtimeseries
```

```
##             Jan       Feb       Mar       Apr       May       Jun       Jul
## 1987   1664.81   2397.53   2840.71   3547.29   3752.96   3714.74   4349.61
## 1988   2499.81   5198.24   7225.14   4806.03   5900.88   4951.34   6179.12
## 1989   4717.02   5702.63   9957.58   5304.78   6492.43   6630.80   7349.62
```

```
## 1990   5921.10   5814.58  12421.25   6369.77   7609.12   7224.75   8121.22
## 1991   4826.64   6470.23   9638.77   8821.17   8722.37  10209.48  11276.55
## 1992   7615.03   9849.69  14558.40  11587.33   9332.56  13082.09  16732.78
## 1993  10243.24  11266.88  21826.84  17357.33  15997.79  18601.53  26155.15
##            Aug       Sep       Oct       Nov       Dec
## 1987   3566.34   5021.82   6423.48   7600.60  19756.21
## 1988   4752.15   5496.43   5835.10  12600.08  28541.72
## 1989   8176.62   8573.17   9690.50  15151.84  34061.01
## 1990   7979.25   8093.06   8476.70  17914.66  30114.41
## 1991  12552.22  11637.39  13606.89  21822.11  45060.69
## 1992  19888.61  23933.38  25391.35  36024.80  80721.71
## 1993  28586.52  30505.41  30821.33  46634.38 104660.67
```

```
ts.plot(souvenirtimeseries)
```



Transforma los datos de trump y hillary en un objeto ts()

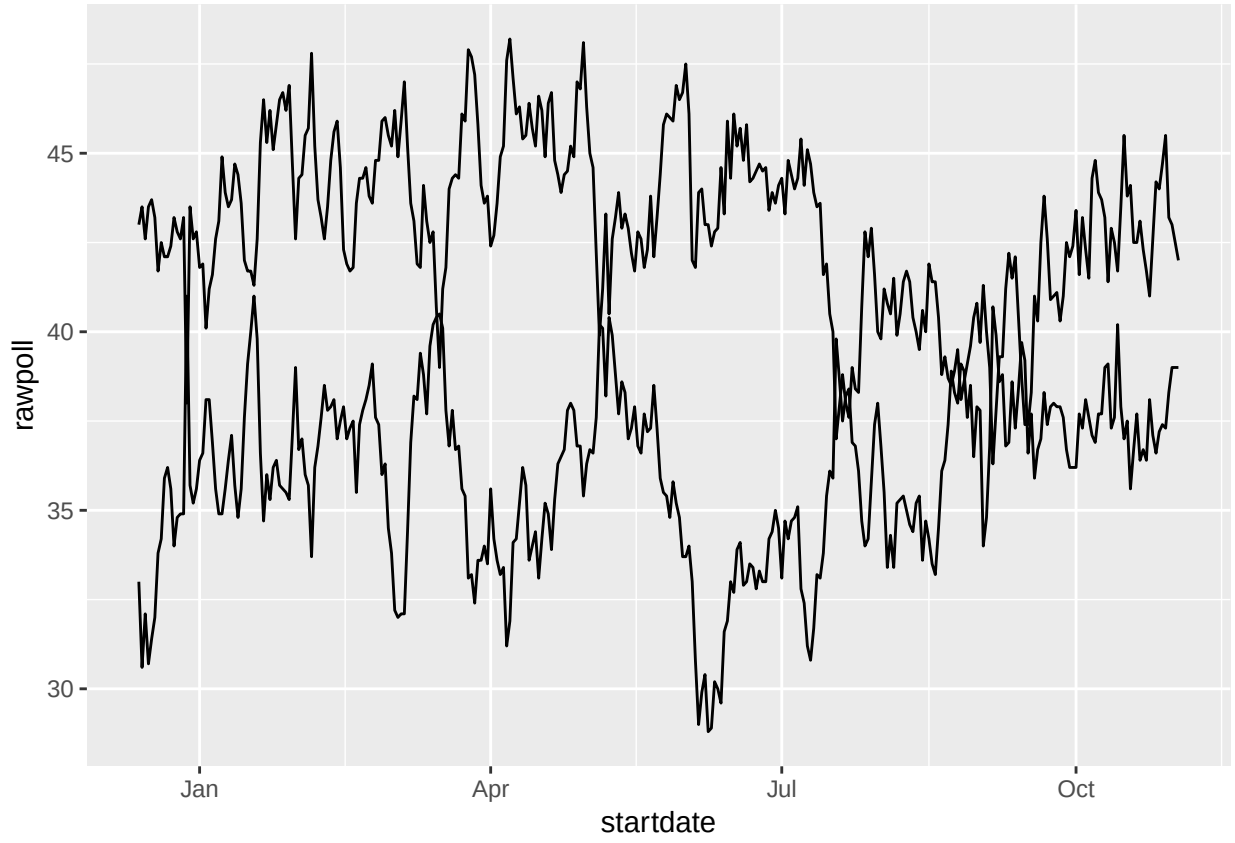
Transforma los datos de las elecciones de US en los polls de trump y clinton

```
data("polls_us_election_2016")
is.ts(polls_us_election_2016)
```

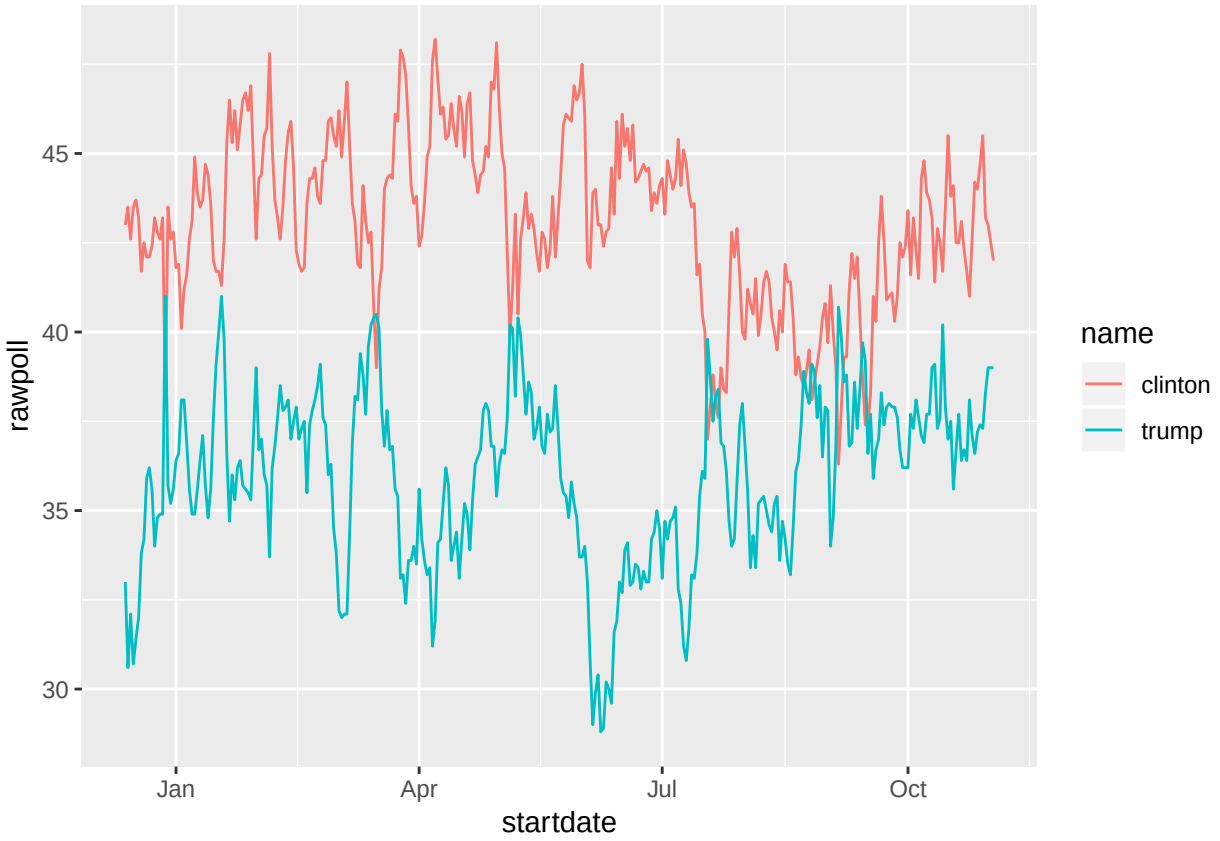
```
## [1] FALSE
```

```
polls_Clinton_ts<-polls_us_election_2016 %>%
                    filter(pollster == "Ipsos" & state =="U.S.") %>%
                    select(startdate,rawpoll_clinton) %>%
                    mutate(rawpoll=rawpoll_clinton) %>% select(startdate,rawpoll)
```

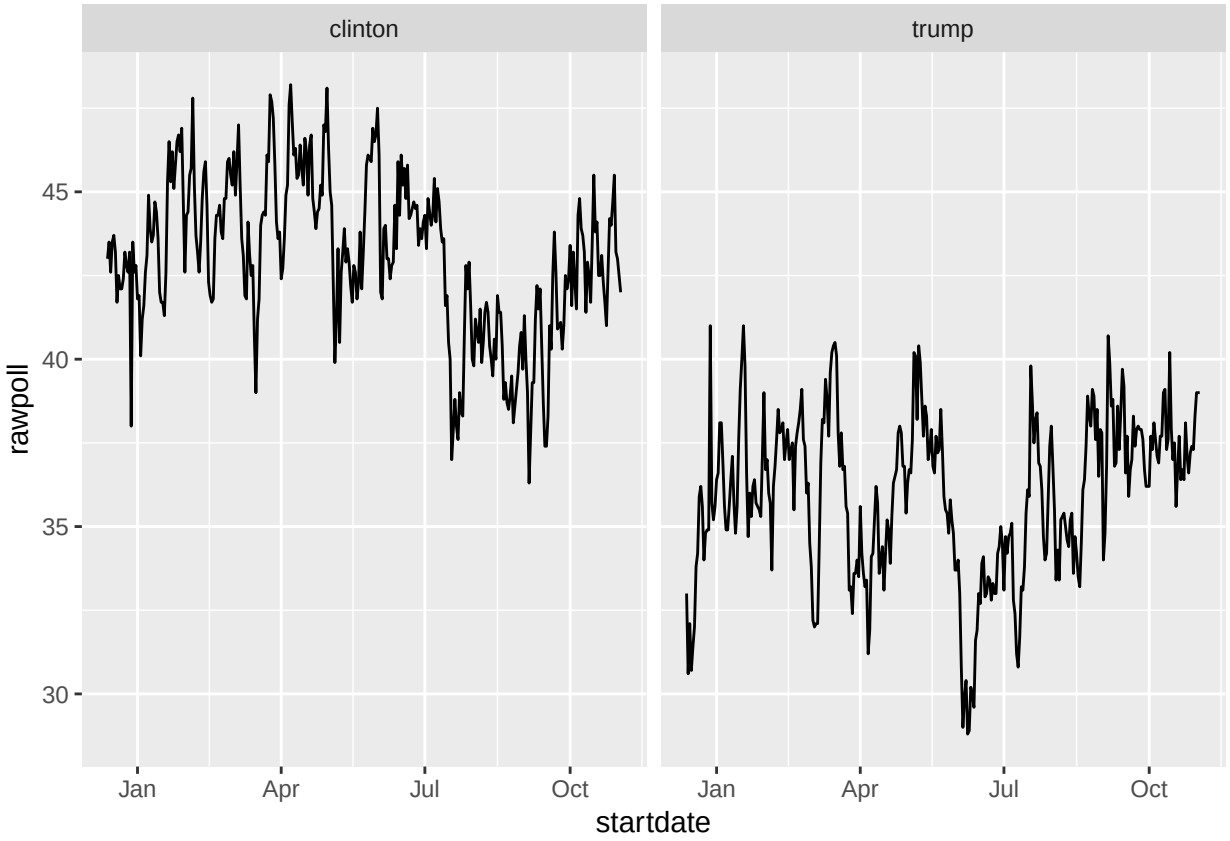
```
polls_trump_ts<-polls_us_election_2016 %>%
  filter(pollster == "Ipsos" & state =="U.S.") %>%
  select(startdate,rawpoll_trump)%>%
  mutate(rawpoll=rawpoll_trump) %>% select(startdate,rawpoll)


polls_both<-bind_rows(polls_Clinton_ts,polls_trump_ts) %>%
              mutate(name=c(rep("clinton",length(polls_Clinton_ts$rawpoll)),
                            rep("trump",length(polls_trump_ts$rawpoll))))

ggplot(polls_both,aes(startdate, rawpoll,group=name)) +geom_line()
```




```
ggplot(polls_both,aes(startdate, rawpoll,color=name)) +geom_line()
```



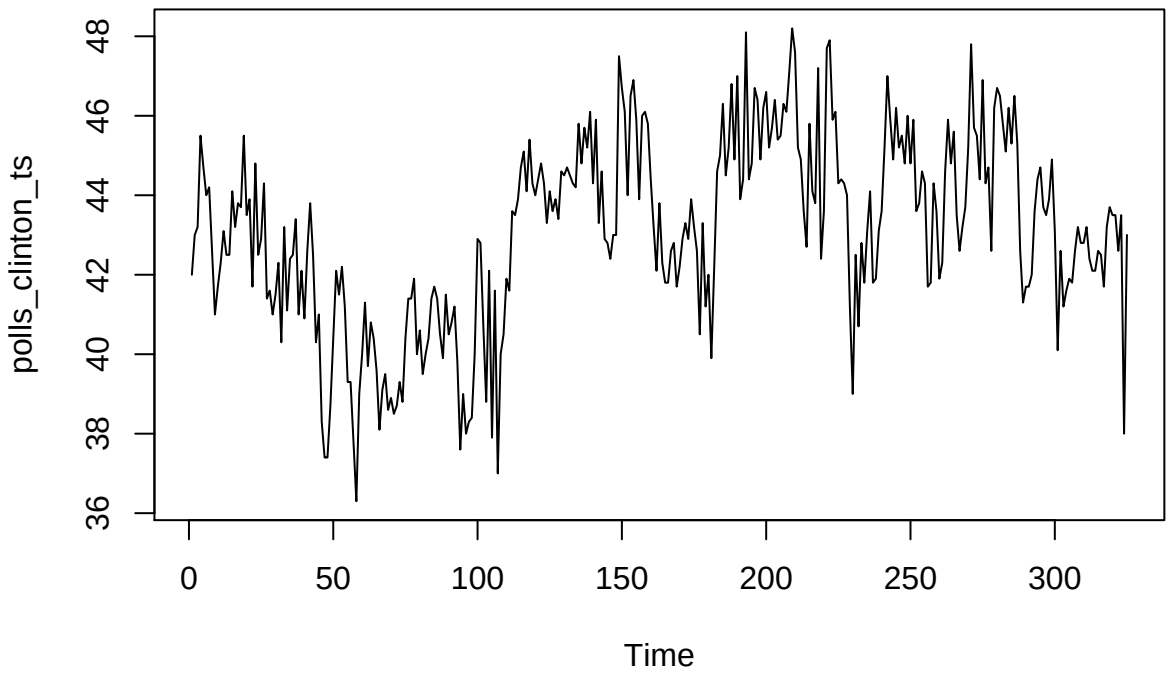

```
ggplot(data = polls_both, mapping = aes(startdate, rawpoll)) +
  geom_line() +
  facet_wrap(~ name)
```



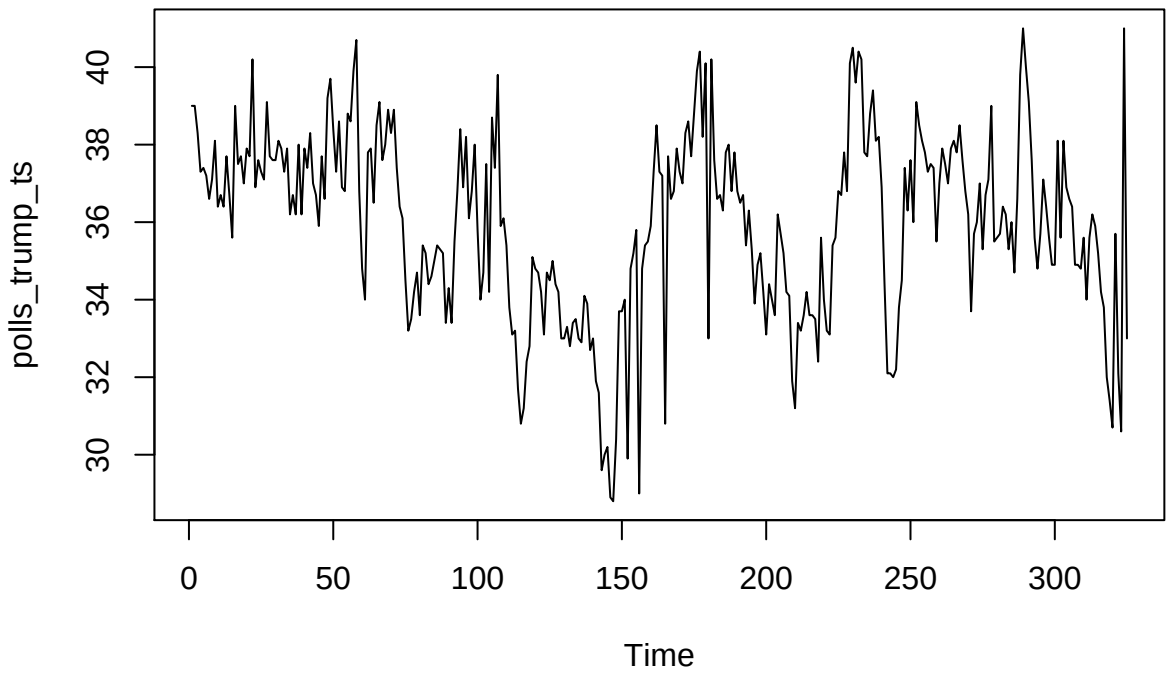

```
head(ymd(polls_both$startdate)-min(ymd(polls_both$startdate)))
```

```
## Time differences in days
## [1] 325 323 322 321 320 319
```

```
# no es una frecuencia regular, por lo tanto no podemos convertirlo en una serie temporal
# el objeto ts() precisa de: datos, fecha de inicio y frecuencia.
# Nos imaginamos que sí que se han medido cada dia para poder trabajar con ello:

polls_clinton_ts=ts(polls_both$rawpoll[which(polls_both$name=="clinton")], frequency = 1, start=1)
ts.plot(polls_clinton_ts)
```




```
polls_trump_ts=ts(polls_both$rawpoll[which(polls_both$name=="trump")], start = 1, frequency = 1)
ts.plot(polls_trump_ts)
```



Análisis descriptivo de las series temporales

Componentes de una serie temporal

Toda serie temporal consta de tres componentes:

- Estacionalidad: Muchas series temporales presentan cierta variación periodica (anual, mensual . . .). Ejemplos de fenómenos estacionales: venta de helados, ocupación hotelera. . .
- Tendencia: Se puede definir como un cambio a largo plazo que se produce en relación al nivel medio, o el cambio a largo plazo de la media. La tendencia se identifica con un movimiento suave de la serie a largo plazo
- Ruido, que se distribuye como una N(0,σ^2) (ruido blanco)

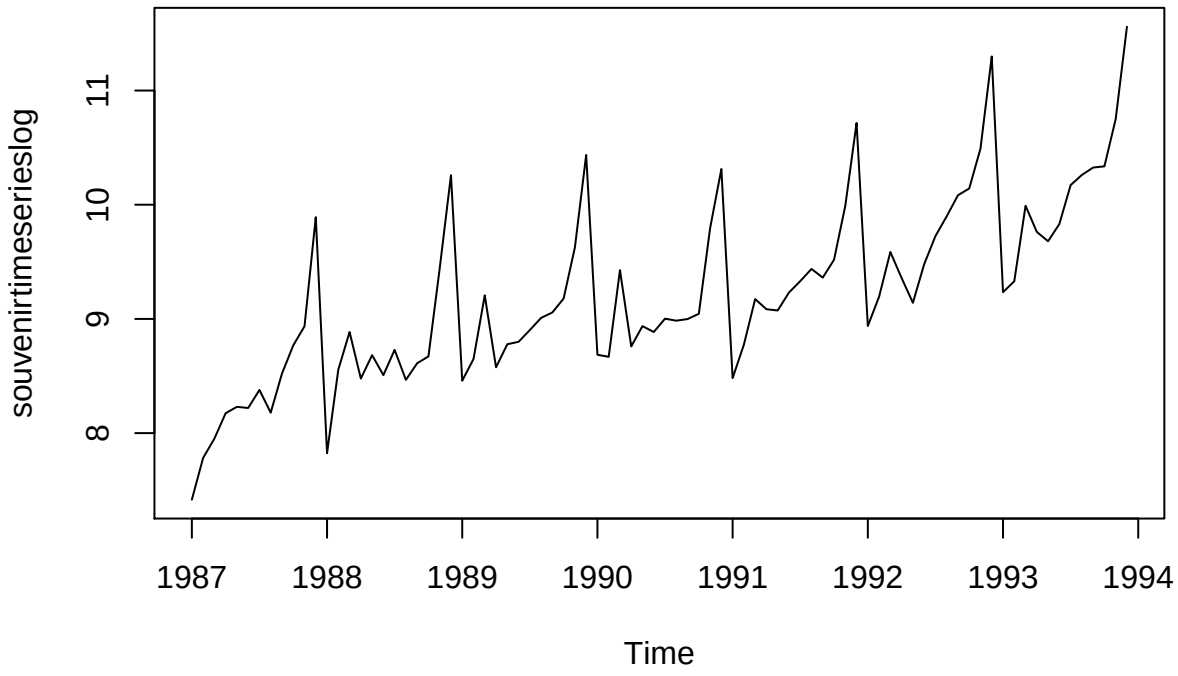
Las series temporales pueden ser

- Aditivas:
$$Y_t = S_t + T_t + \epsilon_t$$
- Multiplicativas
$$Y_t = S_t T_t \epsilon_t$$

Se pasa de una multiplicativa a una aditiva tomando logaritmos.

En los tres ejemplos anteriores, la primera y la segunda serie temporal tienen un ruido que se suma a la tendencia media de la serie. Es por ello una serie temporal aditiva. Sin embargo, en la tercera el ruido es cada vez mayor, se trataría de una serie temporal multiplicativa

```
souvenirtimeserieslog<- ts(log(souvenir), frequency=12, start=c(1987,1))
ts.plot(souvenirtimeserieslog)
```



Clasificación de las series temporales:

- Discrete o continua: dependiendo de la naturaleza de las observaciones. El número de nacimientos por trimestre en NY es discreta mientras que las ventas en la tienda de souvenirs si se expresan en € obtenidos seria continua
- Determinística o estocástica: Si se pueden hacer predicciones con total certeza o si hay un componente aleatorio que nos lo impide
- Según las características de su media y varianza:

++ Estacionaria: Una serie temporal estacionaria es aquella que tiene unas media y varianza constantes en el tiempo. Las edades de las muertes de los reyes sería una serie temporal estacionaria sin componente de estacionalidad.

++ No estacionaria: La media o la varianza cambian con el tiempo. El dataset de los nacimientos y el de la venta de la tienda de souvenirs son ambos no-estacionarios. De hecho son series temporales no estacionarias con componente estacional.

Para comprobar si una serie temporal es o no estacionaria usamos el Augmented Dickey-Fuller Test (adf test).

```
library(tseries)
```

```
## Warning: package 'tseries' was built under R version 3.5.2
```

```r
adf.test(kingstimeseries) # p-value < 0.05 indicates the TS is stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  kingstimeseries
## Dickey-Fuller = -2.1132, Lag order = 3, p-value = 0.529
## alternative hypothesis: stationary
```

```r
kpss.test(kingstimeseries)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  kingstimeseries
## KPSS Level = 0.49131, Truncation lag parameter = 3, p-value =
## 0.04362
```

```r
adf.test(birthstimeseries) # p-value < 0.05 indicates the TS is stationary
```

```
## Warning in adf.test(birthstimeseries): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  birthstimeseries
## Dickey-Fuller = -5.9547, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```r
kpss.test(birthstimeseries)
```

```
## Warning in kpss.test(birthstimeseries): p-value smaller than printed p-
## value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  birthstimeseries
## KPSS Level = 2.7644, Truncation lag parameter = 4, p-value = 0.01
```

```r
adf.test(souvenirtimeserieslog) # p-value < 0.05 indicates the TS is stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  souvenirtimeserieslog
## Dickey-Fuller = -3.7361, Lag order = 4, p-value = 0.02654
## alternative hypothesis: stationary
```

```r
kpss.test(souvenirtimeserieslog)
```

```
## Warning in kpss.test(souvenirtimeserieslog): p-value smaller than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  souvenirtimeserieslog
## KPSS Level = 1.7909, Truncation lag parameter = 3, p-value = 0.01
```

Descomposicion de series temporales

En general, para entender bien el comportamiento de una serie temporal debemos separar sus tres componentes: la tendencia, la componente estacional y el ruido. Si una serie temporal tiene patrones repetidos sin una frecuencia fija y la serie es mayor de 2 años, hablamos de un ciclo.

Descomposición de las series temporales sin componente de estacionalidad

El dataset de los reyes lo ilustra.

- Smoothing

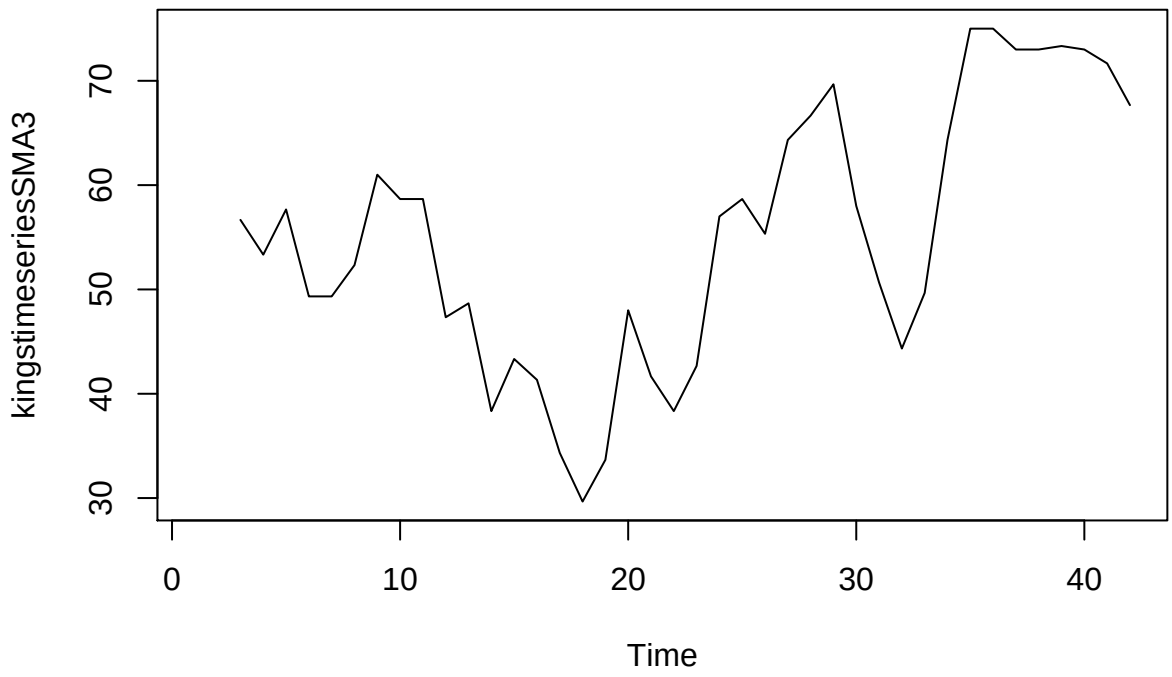
Como ya hemos dicho, la tendencia viene a ser la evolución de la serie temporal que veríamos si no hubiera componente de estacionalidad. Una forma de conseguirlo es considerando, no todos los puntos, sino las *runningmeans* para ventanas del tamaño que consideremos. Esto se denomina *smoothing*

En el ejemplo de los reyes vamos tomando mediciones de 3 en 3.

```
library(TTR)
```

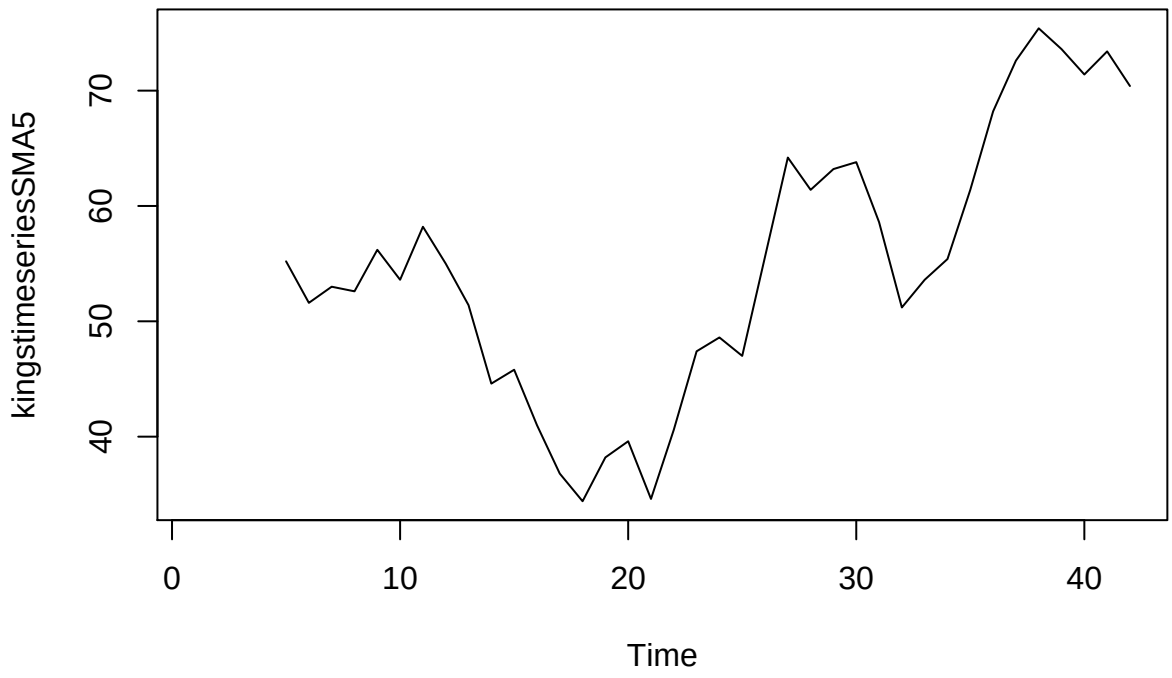
```
## Warning: package 'TTR' was built under R version 3.5.2
```

```
kingstimeseriesSMA3 <- SMA(kingstimeseries,n=3)
plot.ts(kingstimeseriesSMA3)
```



Y vamos cambiando el tamaño de la ventana de medias:

```
kingstimeseriesSMA5 <- SMA(kingstimeseries,n=5)
plot.ts(kingstimeseriesSMA5)
```



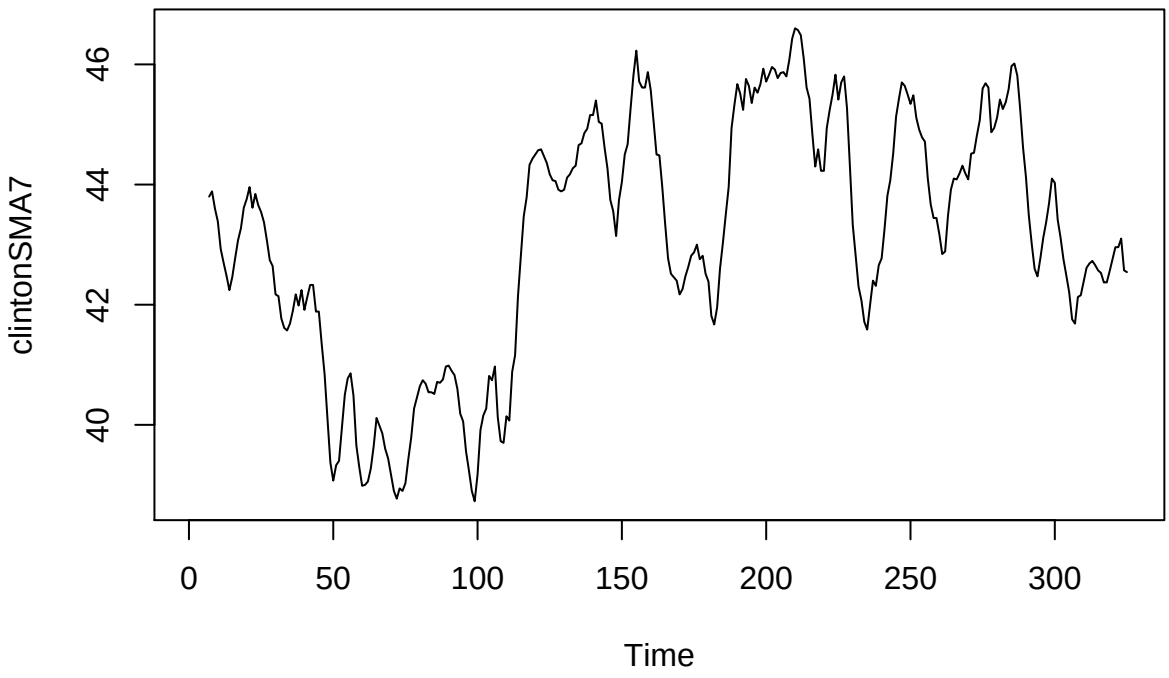

```
kingstimeseriesSMA14 <- SMA(kingstimeseries,n=14)
plot.ts(kingstimeseriesSMA14)
```

Lo cual nos da una idea mucho más clara de cómo fue disminuyendo hasta alrededor del rey 25 para despues ir aumentando hasta el final del reino.

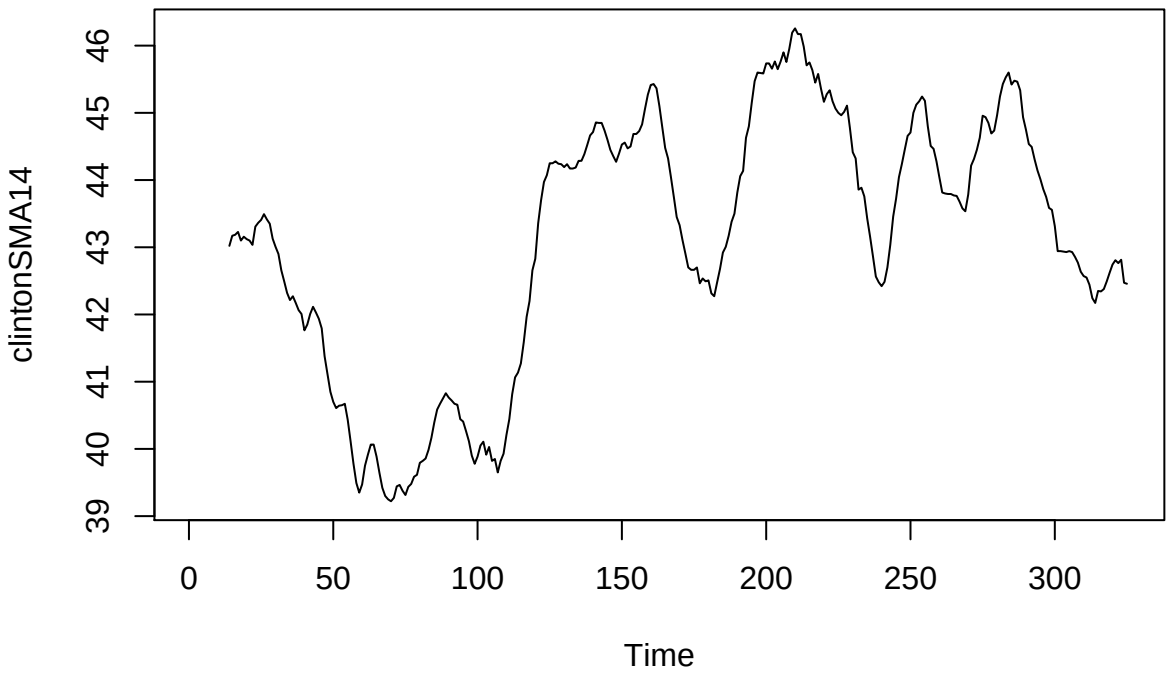
Ejercicicio: Trump vs Clinton

Utiliza smoothing para saber como fueron las tendencias del voto a Hillary durante el periodo electoral:

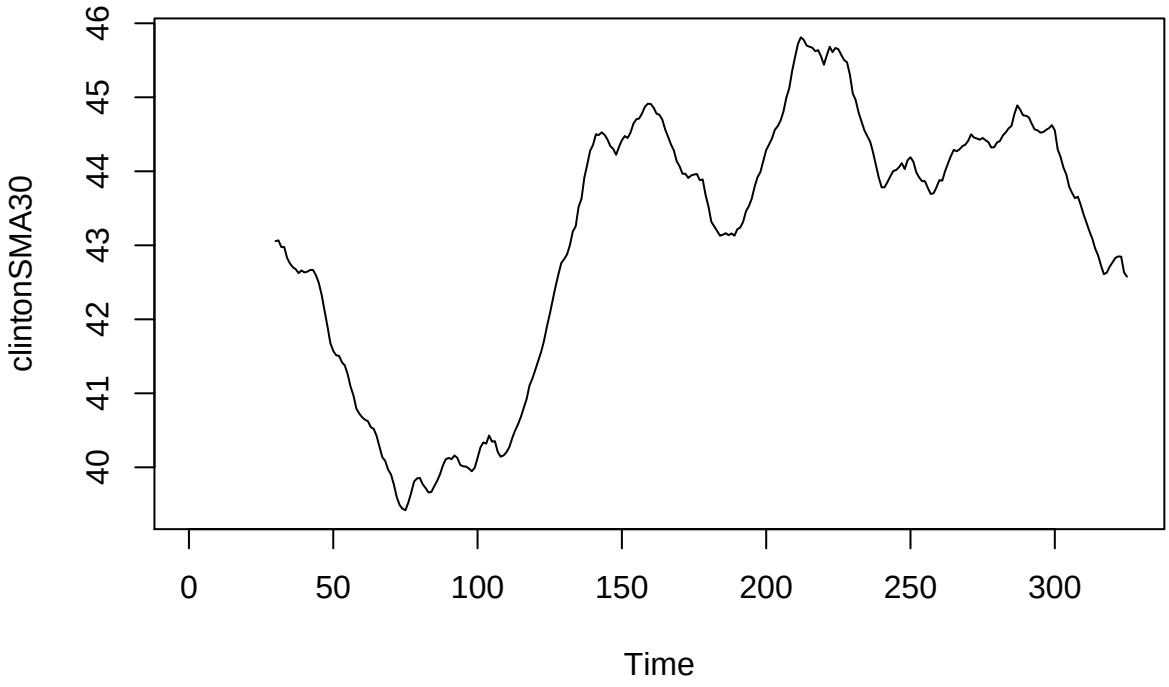
```
clintonSMA7 <- SMA(polls_clinton_ts,n=7)
plot.ts(clintonSMA7)
```



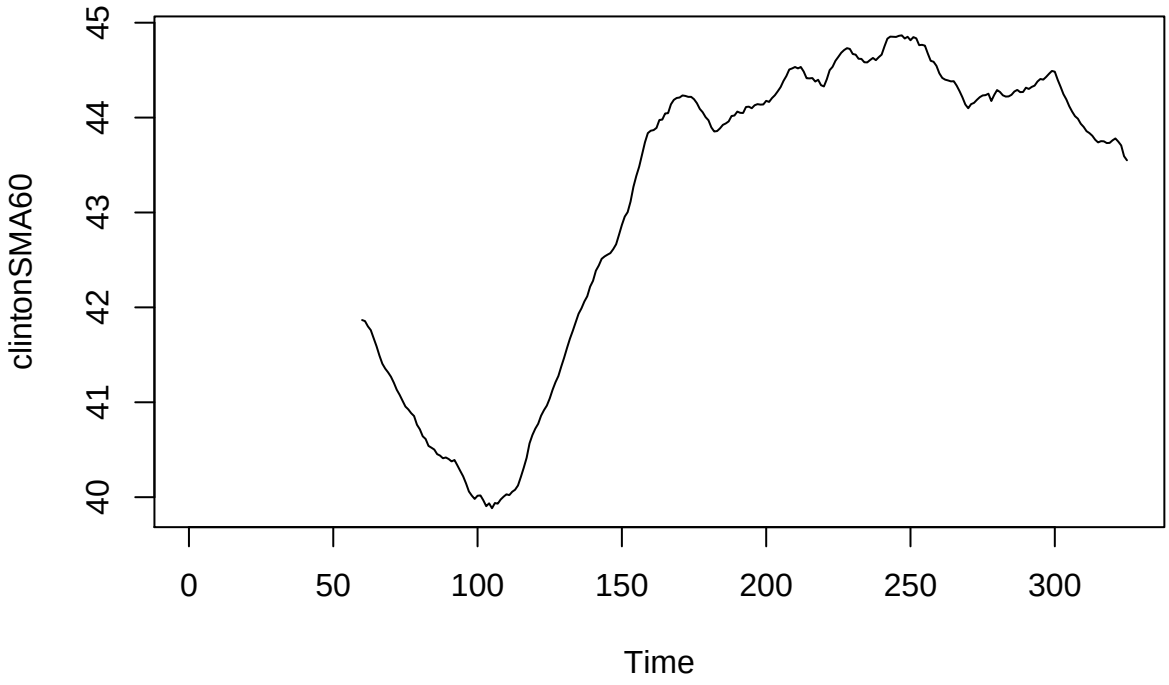

```
clintonSMA14 <- SMA(polls_clinton_ts,n=14)
plot.ts(clintonSMA14)
```




```
clintonSMA30 <- SMA(polls_clinton_ts,n=30)
plot.ts(clintonSMA30)
```

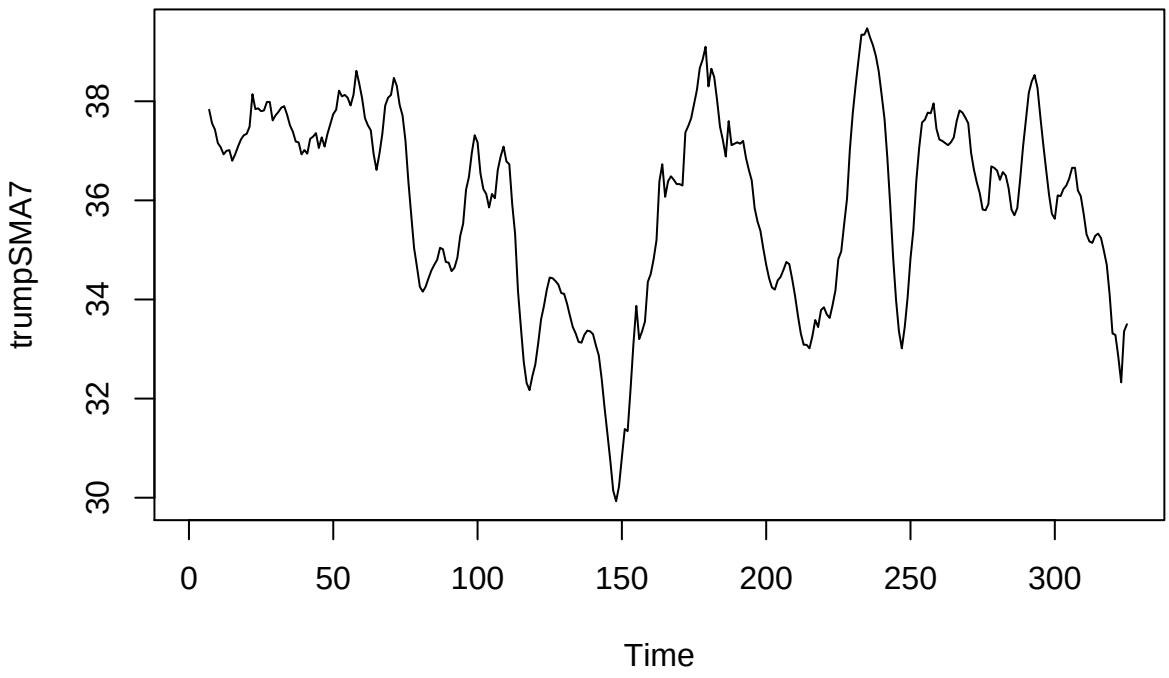



```
clintonSMA60 <- SMA(polls_clinton_ts,n=60)
plot.ts(clintonSMA60)
```

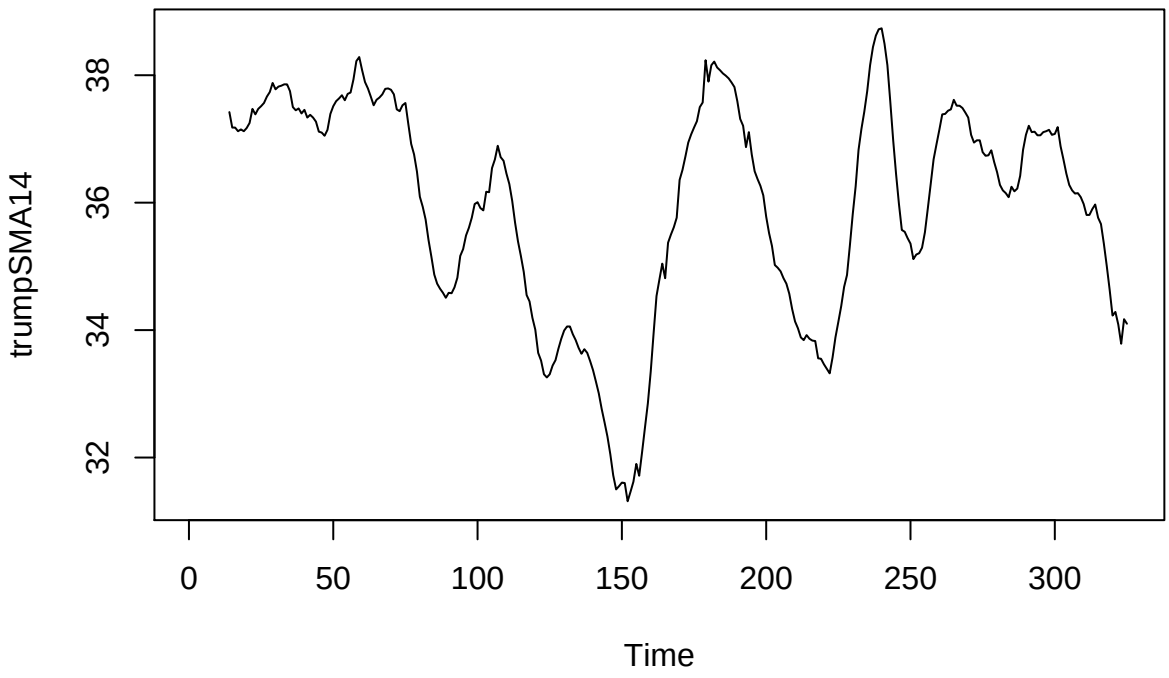


Y a trump?

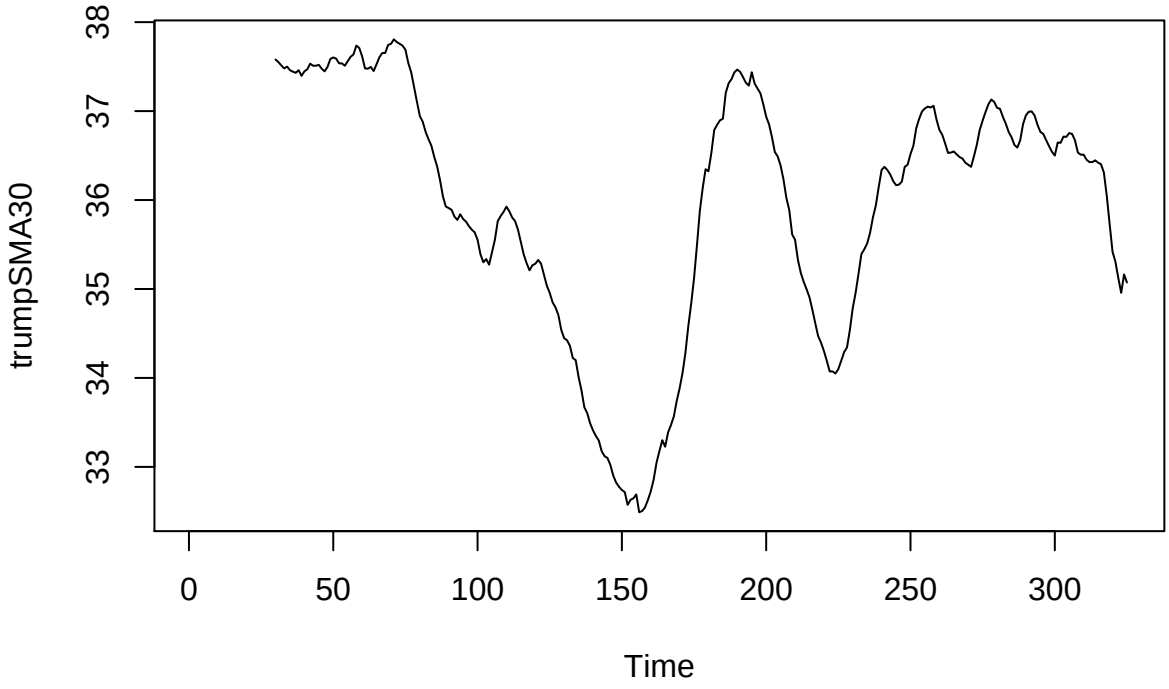
```
trumpSMA7 <- SMA(polls_trump_ts,n=7)
plot.ts(trumpSMA7)
```



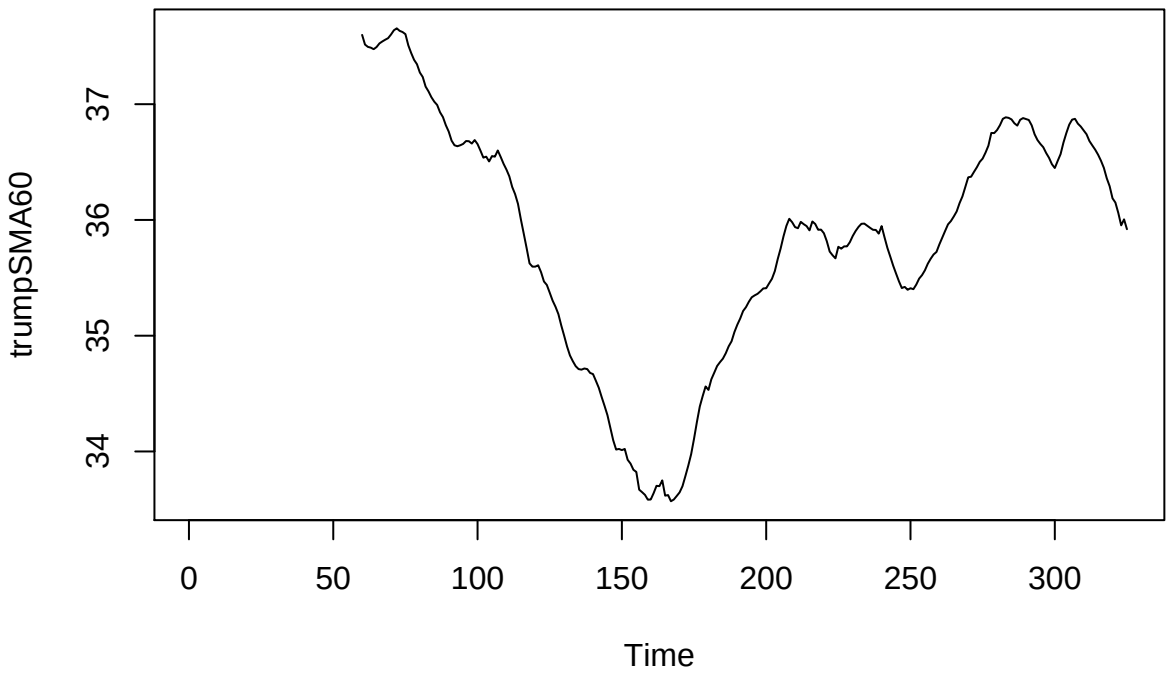

```
trumpSMA14 <- SMA(polls_trump_ts,n=14)
plot.ts(trumpSMA14)
```



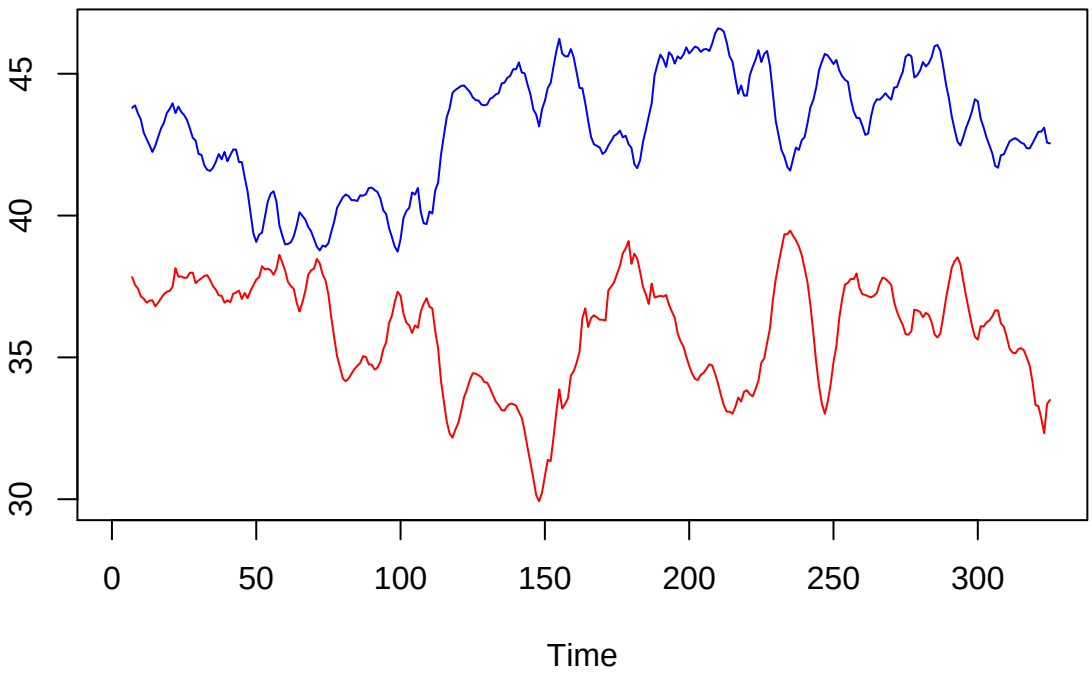

```
trumpSMA30 <- SMA(polls_trump_ts,n=30)
plot.ts(trumpSMA30)
```



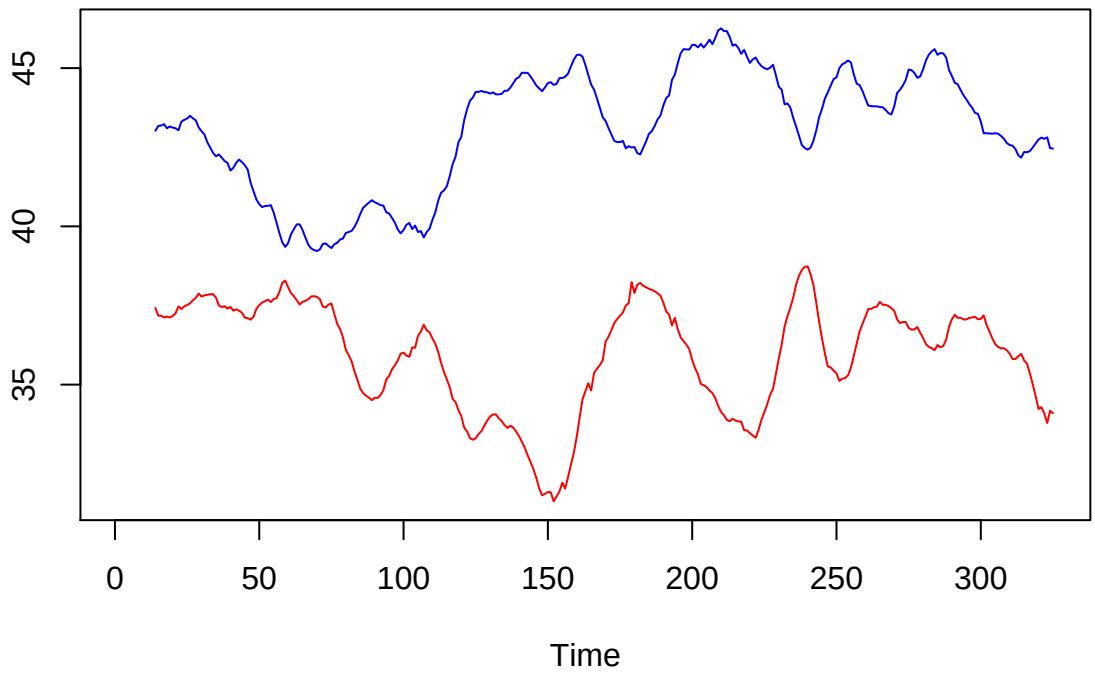

```
trumpSMA60 <- SMA(polls_trump_ts,n=60)
plot.ts(trumpSMA60)
```



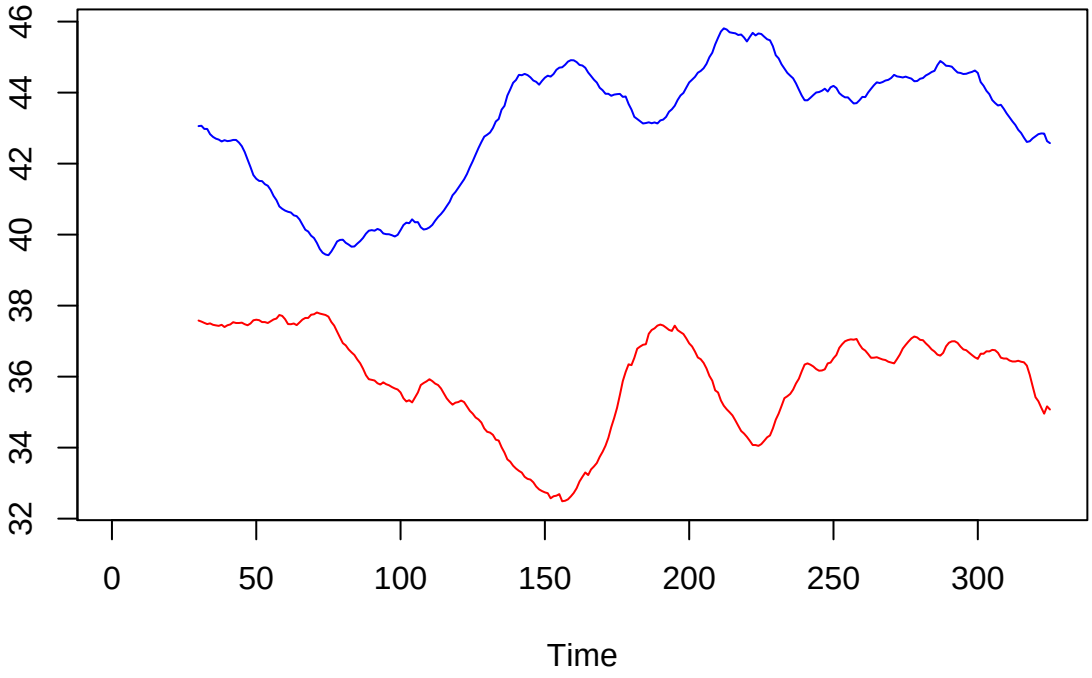

```
ts.plot(trumpSMA7,clintonSMA7,gpars = list(col = c("red", "blue")))
```



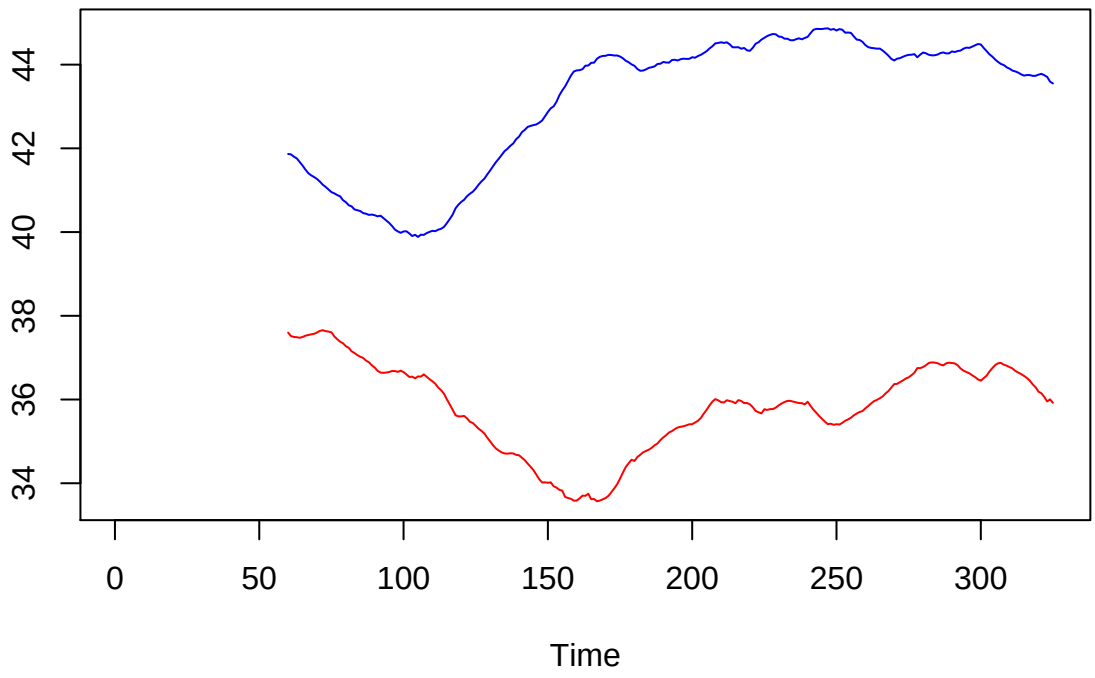

```
ts.plot(trumpSMA14,clintonSMA14,gpars = list(col = c("red", "blue")))
```




```
ts.plot(trumpSMA30,clintonSMA30,gpars = list(col = c("red", "blue")))
```



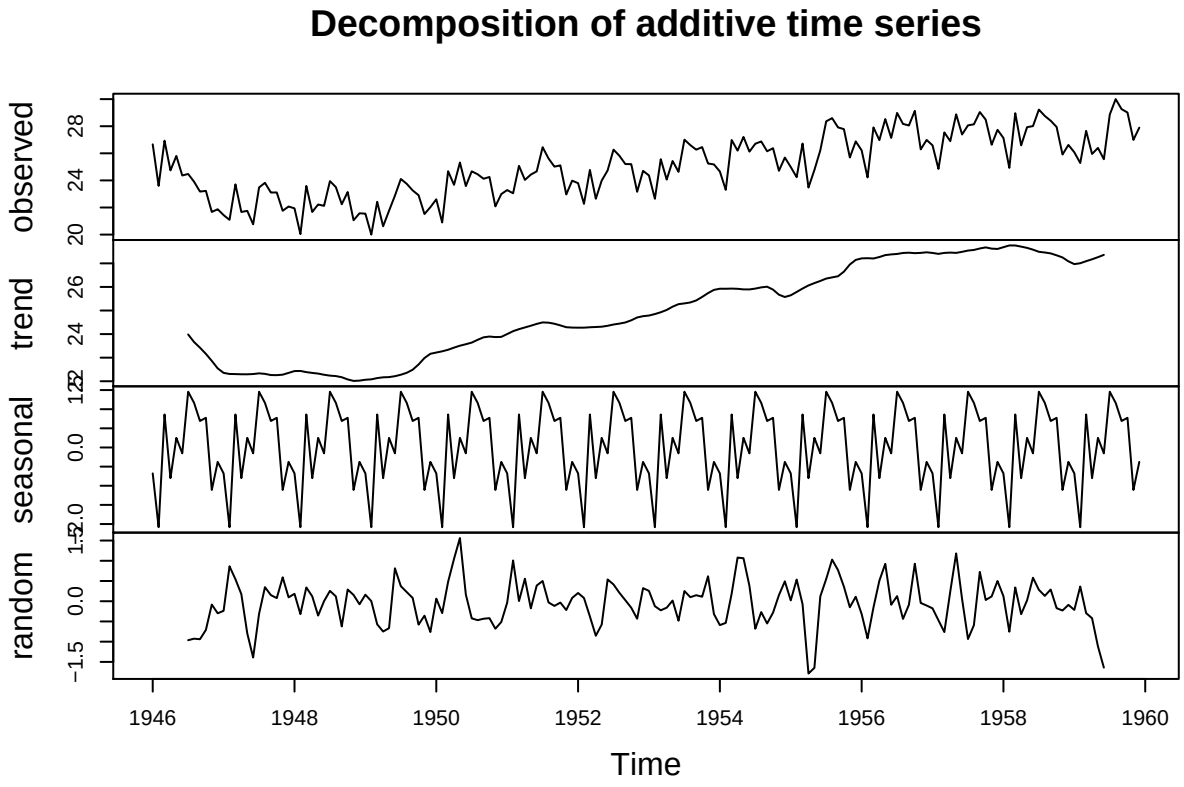

```
ts.plot(trumpSMA60,clintonSMA60,gpars = list(col = c("red", "blue")))
```



Descomposición de las series temporales con componente de estacionalidad

Si la serie presenta algún patron periódico, una simple media móvil (*runningmean*) puede ayudar. Así lo hace la función *decompose* que permite hacer una descomposicion para una serie aditiva o multiplicativa.

```
birthstimeseriescomponents <- decompose(birthstimeseries)
plot(birthstimeseriescomponents)
```



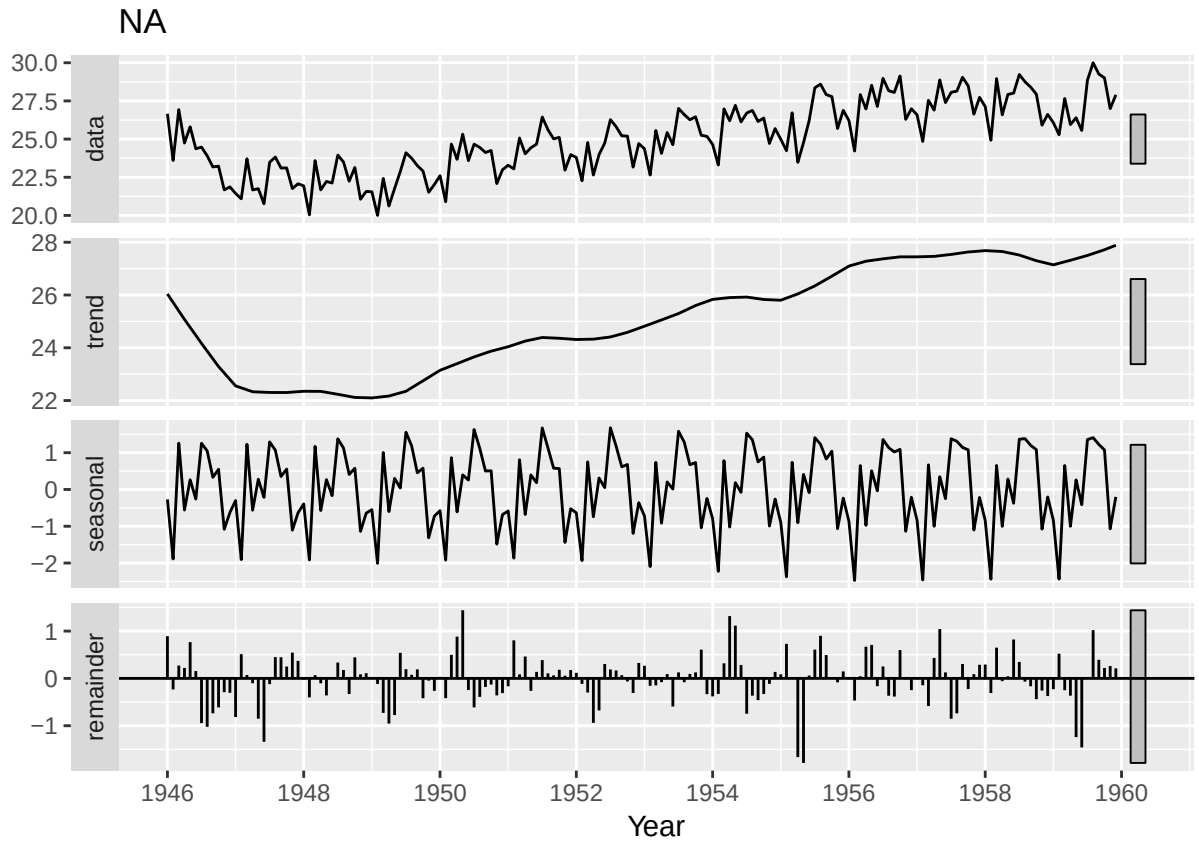
La función decompose determina en primer lugar la componente de tendencia usando una media móvil como en los ejemplos anteriores y la quita de la serie sustrayendola. La componente estacional se calcula con la media para cada unidad de tiempo sobre todos los periodos. Esta media se centra. Finalmente, la componente del error se determina quitando la tendecia y la componente estacional de los datos originales.

Una implementación más sofisticada es *stl*

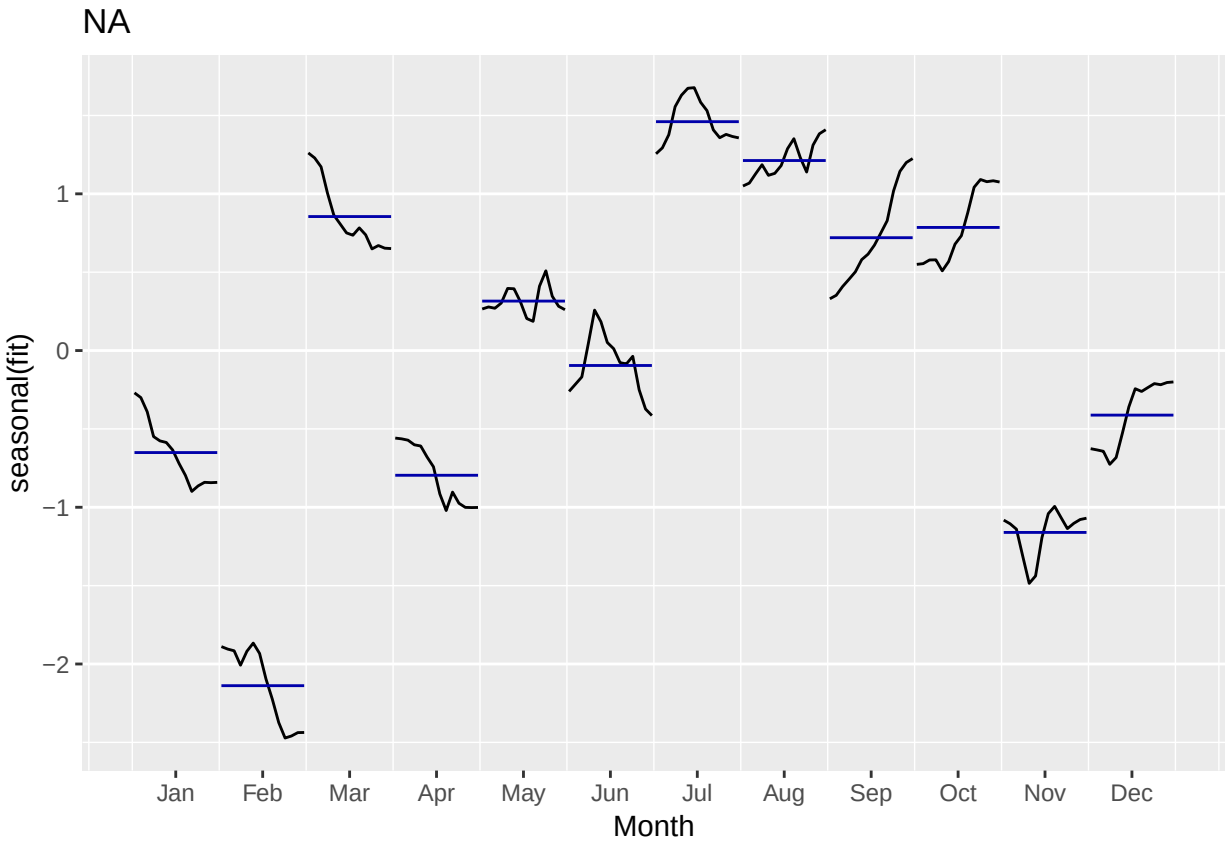
stl (seasonal, trend, irregular)

Para determinar la componente de tendencia este método utiliza un tipo de *smoothing* llamado *loess* que hace una regresión lineal en una ventana de puntos pequeña para descomponer la serie en sus tres partes.

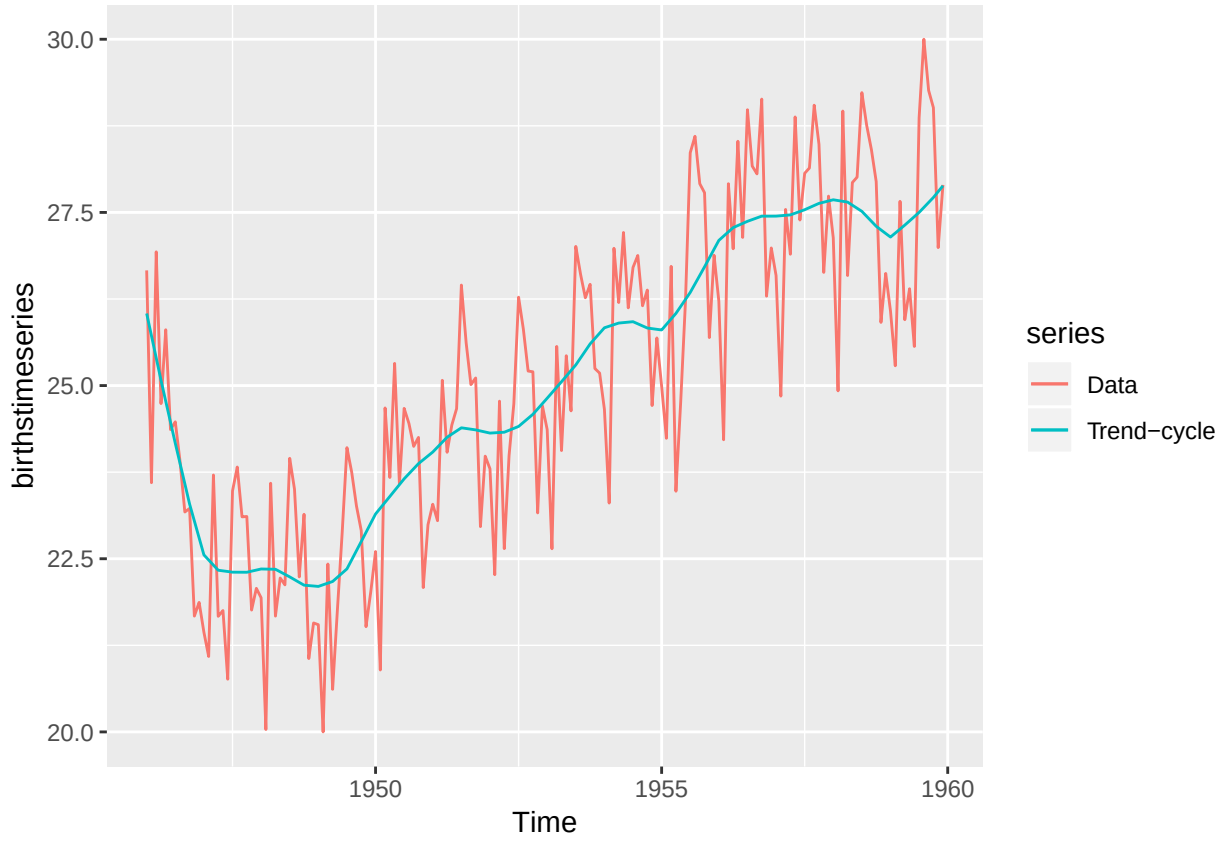
```
fit <- stl(birthstimeseries,s.window=7)
autoplot(fit)+xlab("Year")
```




```
ggsubseriesplot(seasonal(fit))
```




```
autoplot(birthstimeseries, series="Data") +
autolayer(trendcycle(fit), series="Trend-cycle")
```



Historia y uso de los métodos de decomposicion de series temporales

- Método original en 1920s.
- Census II se introdujo en 1957. Fue la base para el método X-11 y sus variantes (X-12-ARIMA, X-13-ARIMA)
- El método STL se introdujo en 1983
- TRAMO/SEATS en los 90s.
- ABS usa X-12-ARIMA
- US Census Bureau:X-13-ARIMA-SEATS
- Statistics Canada: X-12-ARIMA
- ONS (UK): X-12-ARIMA
- EuroStat: X-13-ARIMA-SEATS
- INE: $http://www.ine.es/daco/daco42/daco4214/cbtc30.pdf$

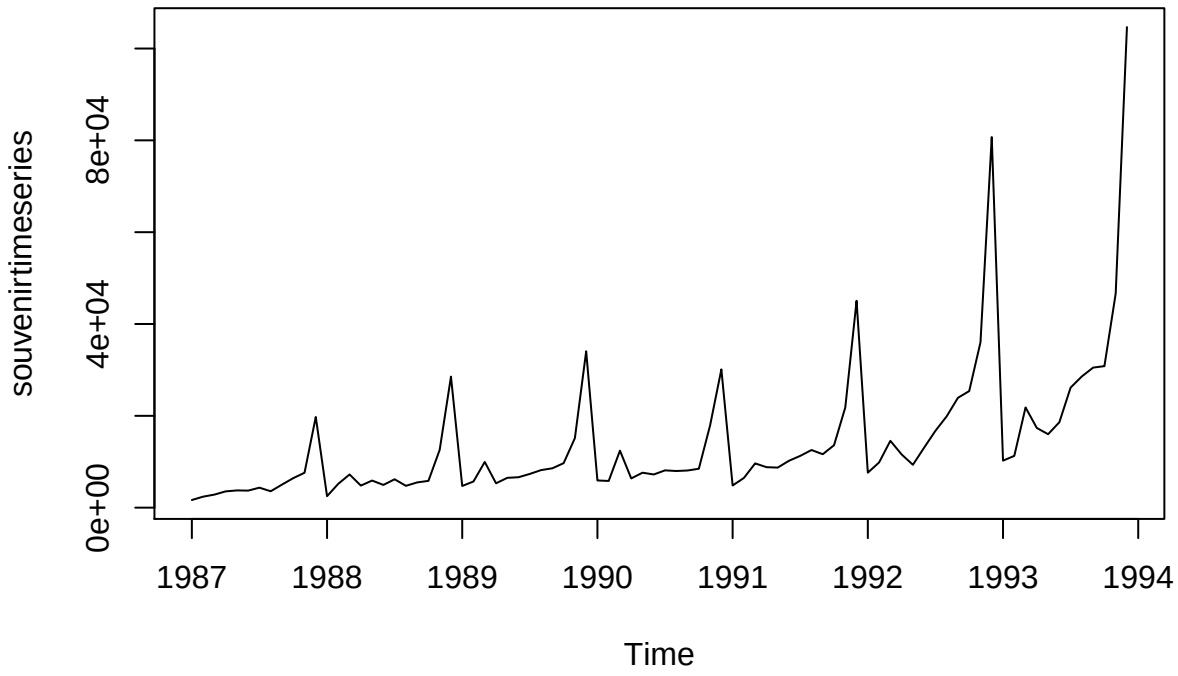
X-11

Se basan en el concepto del ratio a la media móvil (1931 by Fredrick R. Macaulay, of the National Bureau of Economic Research in the US). Básicamente el proceso consiste en 3 pasos:

1) Estimación de la tendencia usando medias móviles
2) Substraccion de la tendencia dejando solo la componente estacional y los errores.
3) Estimación de la componente estacional mediante medias móviles que suavicen los errores.

Como los pasos 1 y 3 se necesitan uno a otro utilizamos un proceso iterativo entre ambos. Asumimos un modelo multiplicativo como el que observábamos en el caso de las ventas de la tienda de souvenirs.

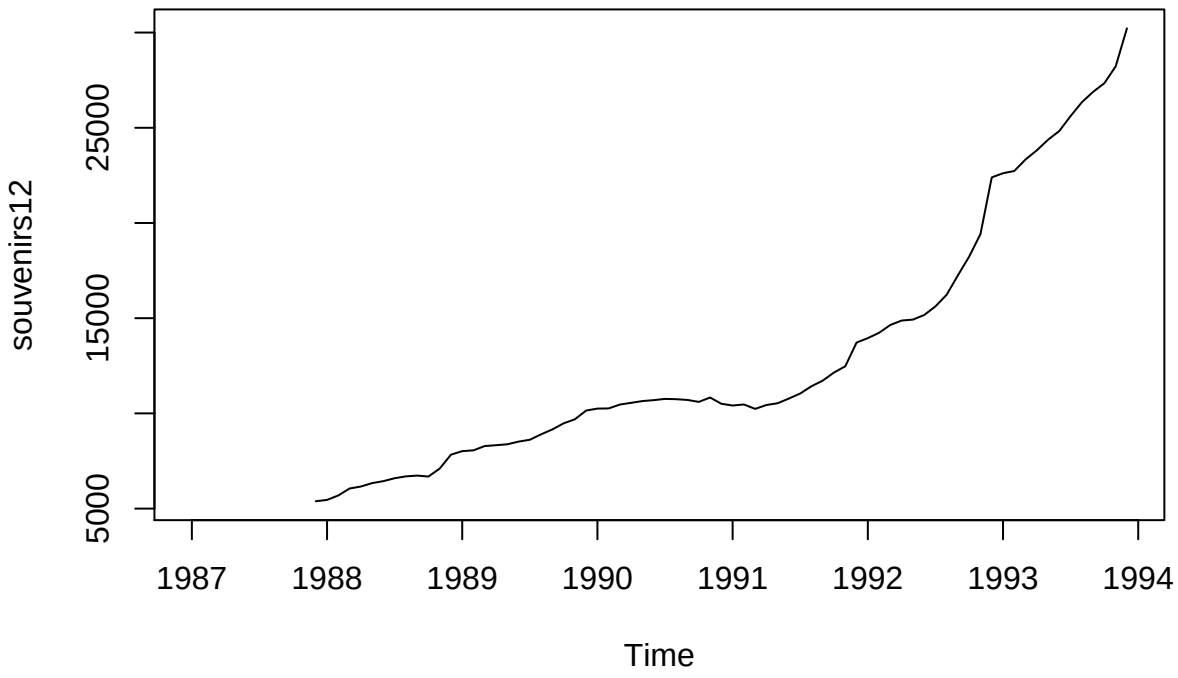
```
souvenir <- scan("http://robjhyndman.com/tsdldata/data/fancy.dat")
souvenirtimeseries <- ts(souvenir, frequency=12, start=c(1987,1))
#souvenirtimeseries
ts.plot(souvenirtimeseries)
```



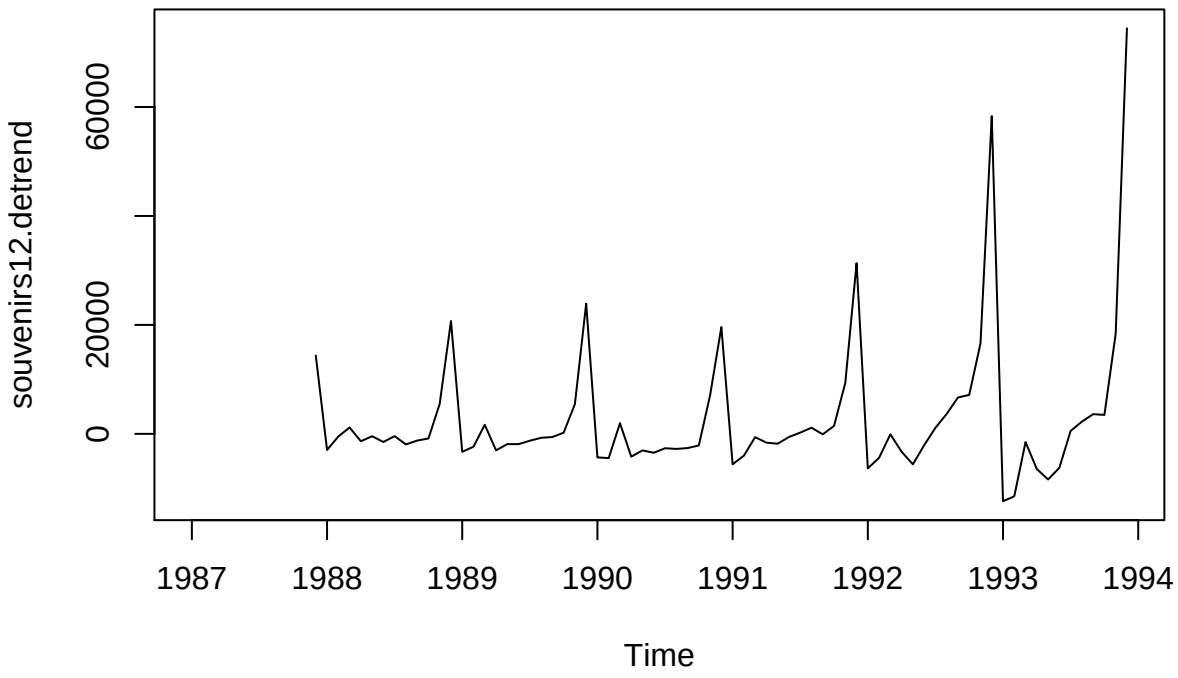
Step 1: Estimador inicial de la tendencia

Aplicar una media móvil de tamaño 12 y quitarla de la serie temporal original:

```
souvenirs12 <- SMA(souvenirtimeseries,n=12)
plot.ts(souvenirs12)
```



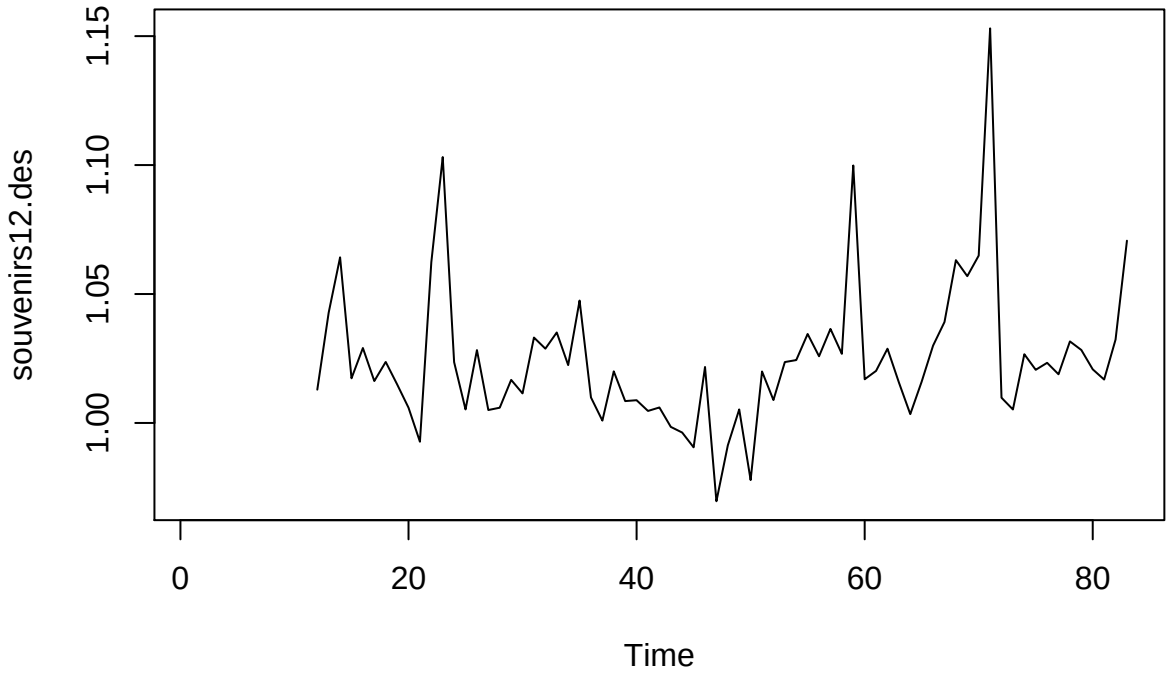

```
souvenirs12.detrend=souvenirtimeseries-souvenirs12
plot.ts(souvenirs12.detrend)
```



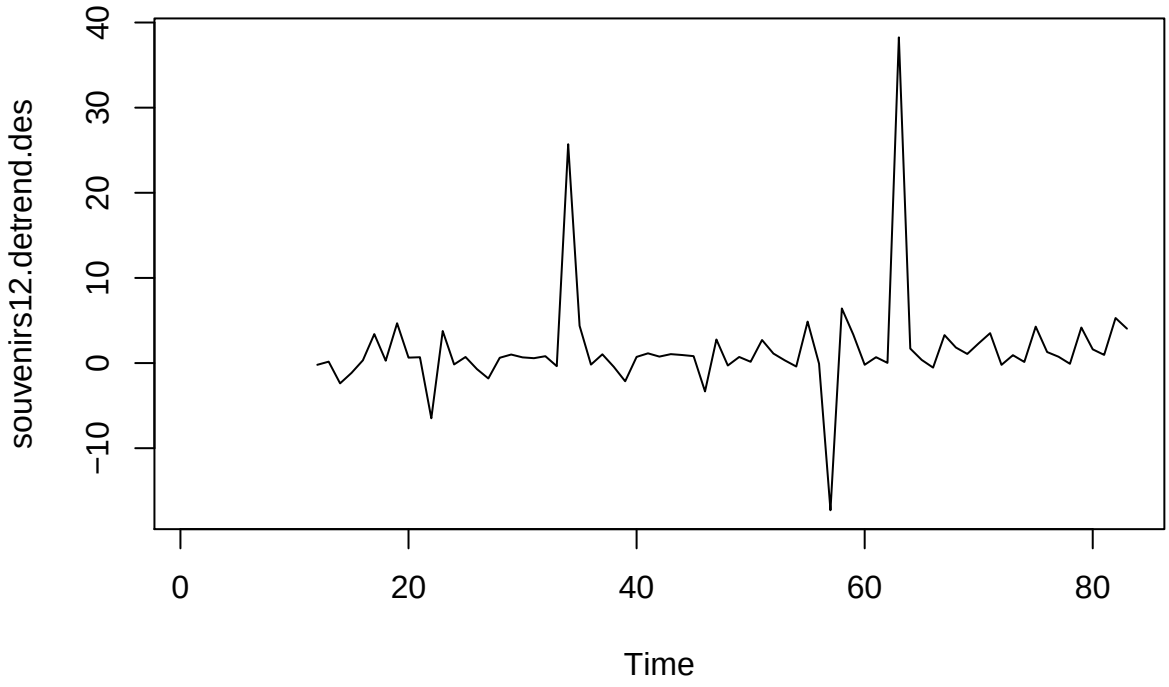
Step 3: Estimador preliminar de la componente estacional de los datos

Dividimos cada punto por su punto anterior:

```
souvenirs12.des=souvenirs12[2:84]/souvenirs12[1:83]
plot.ts(souvenirs12.des)
```




```
souvenirs12.detrend.des=souvenirs12.detrend[2:84]/souvenirs12.detrend[1:83]
plot.ts(souvenirs12.detrend.des)
```



A partir de aqui el proceso se repite iterativamente mejorando la estimacion de la tendencia usando Henderson media móvil y nuevamente el paso 2 y repetimos otra vez este proceso (3 iteraciones). Las versiones de X11 que son X12ARIMA y SEASABS usan versiones específicas de las medias móviles de Henderson.

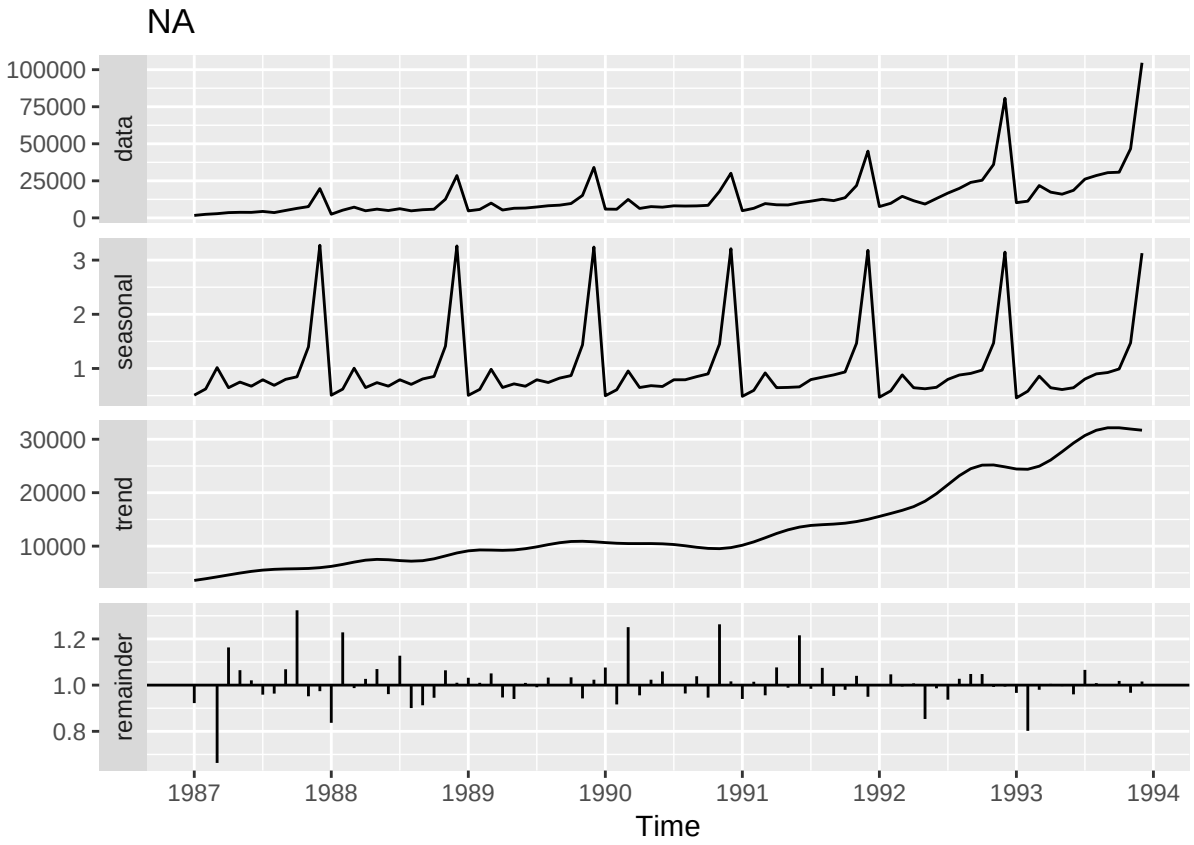
```
library(seasonal)
```

```
## Warning: package 'seasonal' was built under R version 3.5.2
```

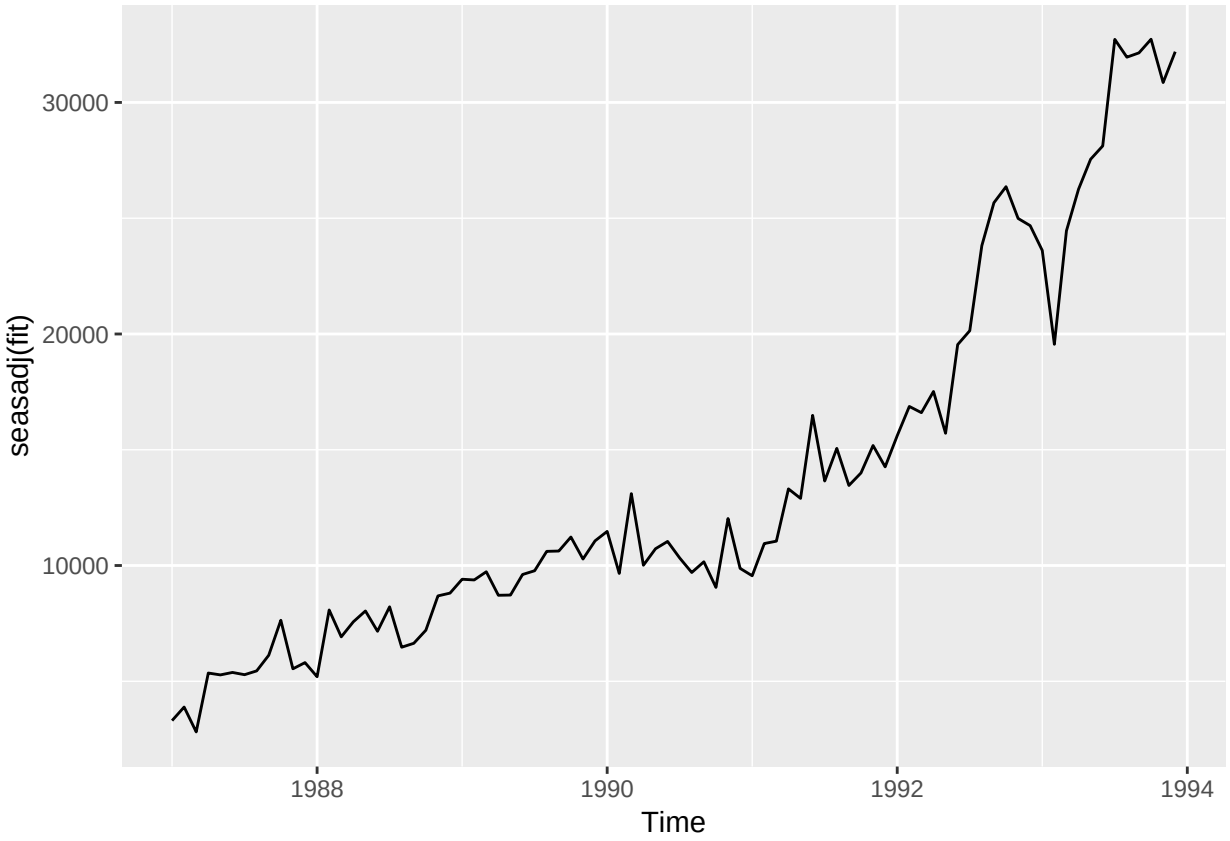
```
##
## Attaching package: 'seasonal'
```

```
## The following object is masked from 'package:tibble':
##
##     view
```

```
fit <- seas(souvenirtimeseries, x11="")
autoplot(fit)
```




```
autoplot(seasadj(fit))
```

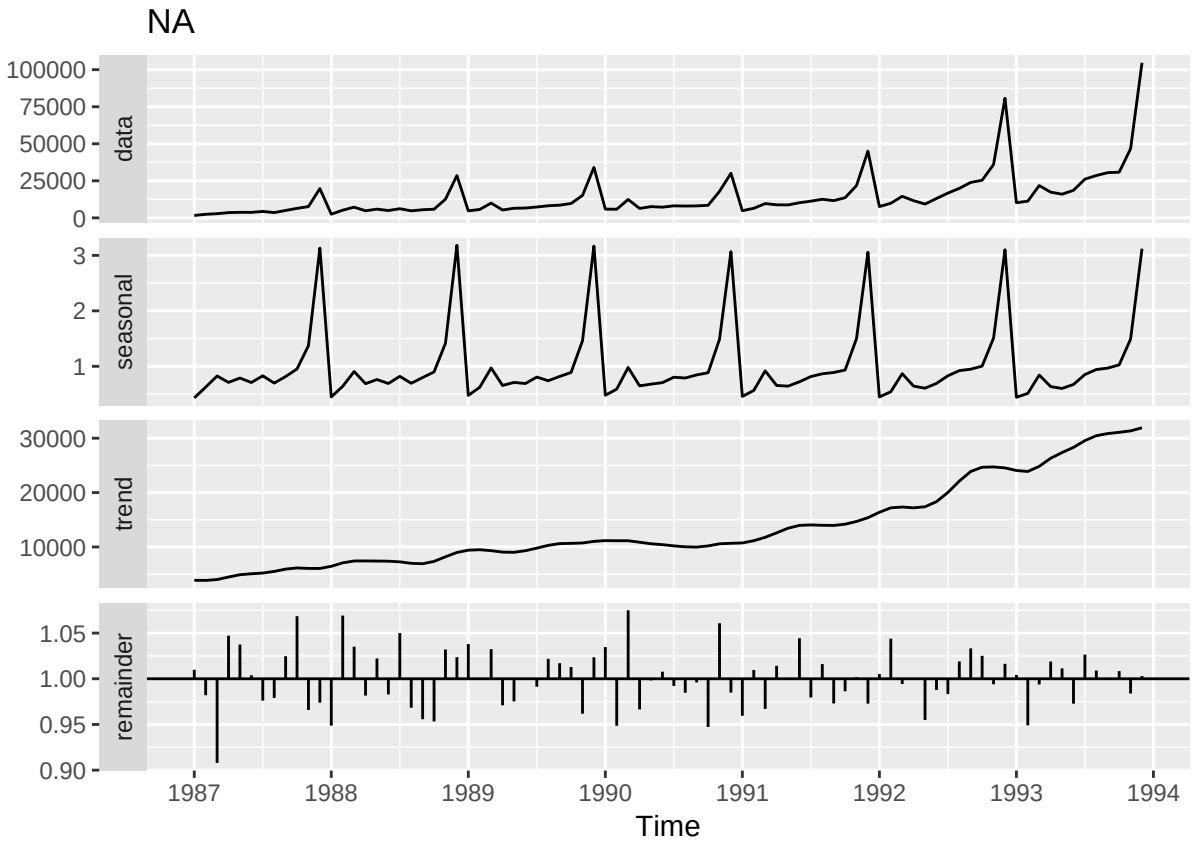


Ventajas: * Robusta a outliers * Muy usada * Las decisiones acerca de la tendencia y los cambios estacionales están muy establecidos

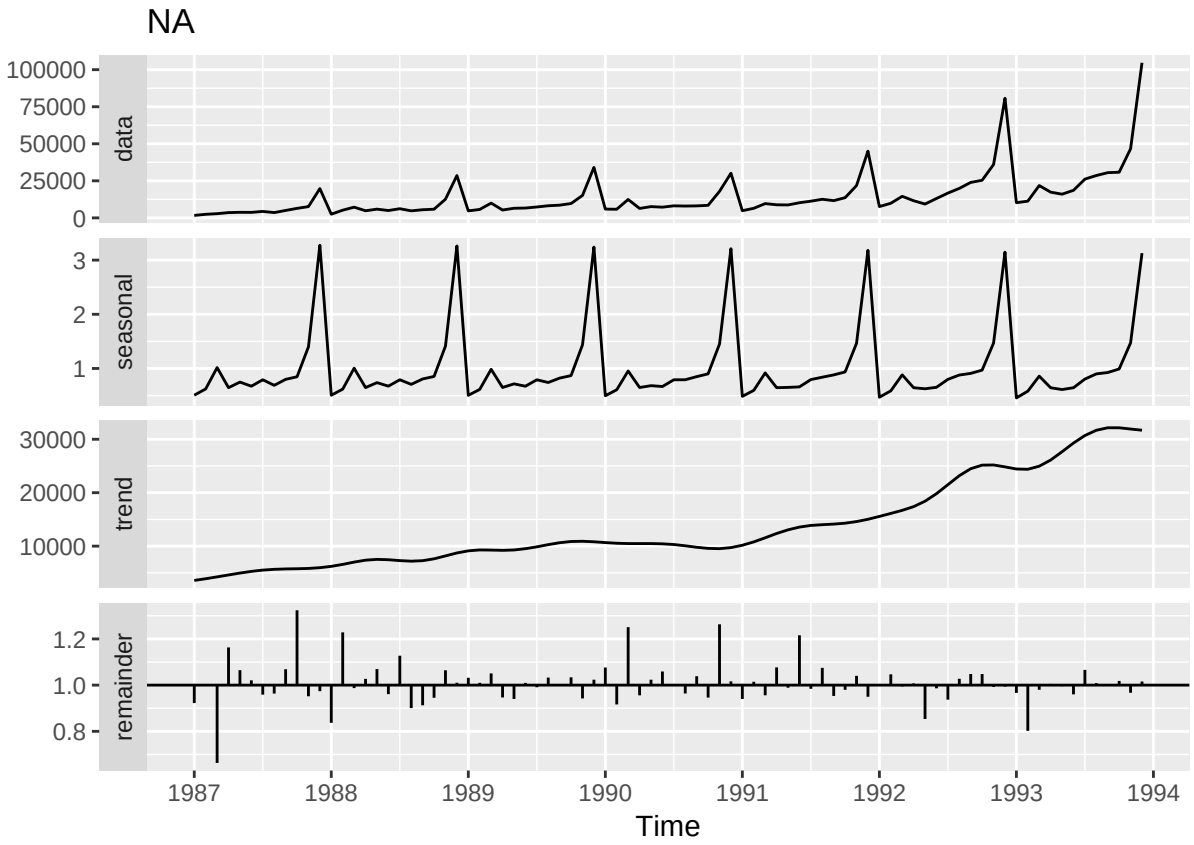
Desventajas: * No da predicciones con probabilidades (CI) No prediction/confidence intervals * Método empírico sin modelo subyacente * Sólo válido para datos cuatrimestral y mensual

SEATS

```
library(seasonal)
fit <- seas(souvenirtimeseries)
autoplot(fit)
```




```
fit2 <- seas(souvenirtimeseries, x11="")
autoplot(fit2)
```



Ventajas: * Model-based * Hace una estimación *loess* de la tendencia * Como usa un modelo puede predecir puntos de inicio y de fin

Desventajas: * Sólo válido para datos cuatrimestral y mensual

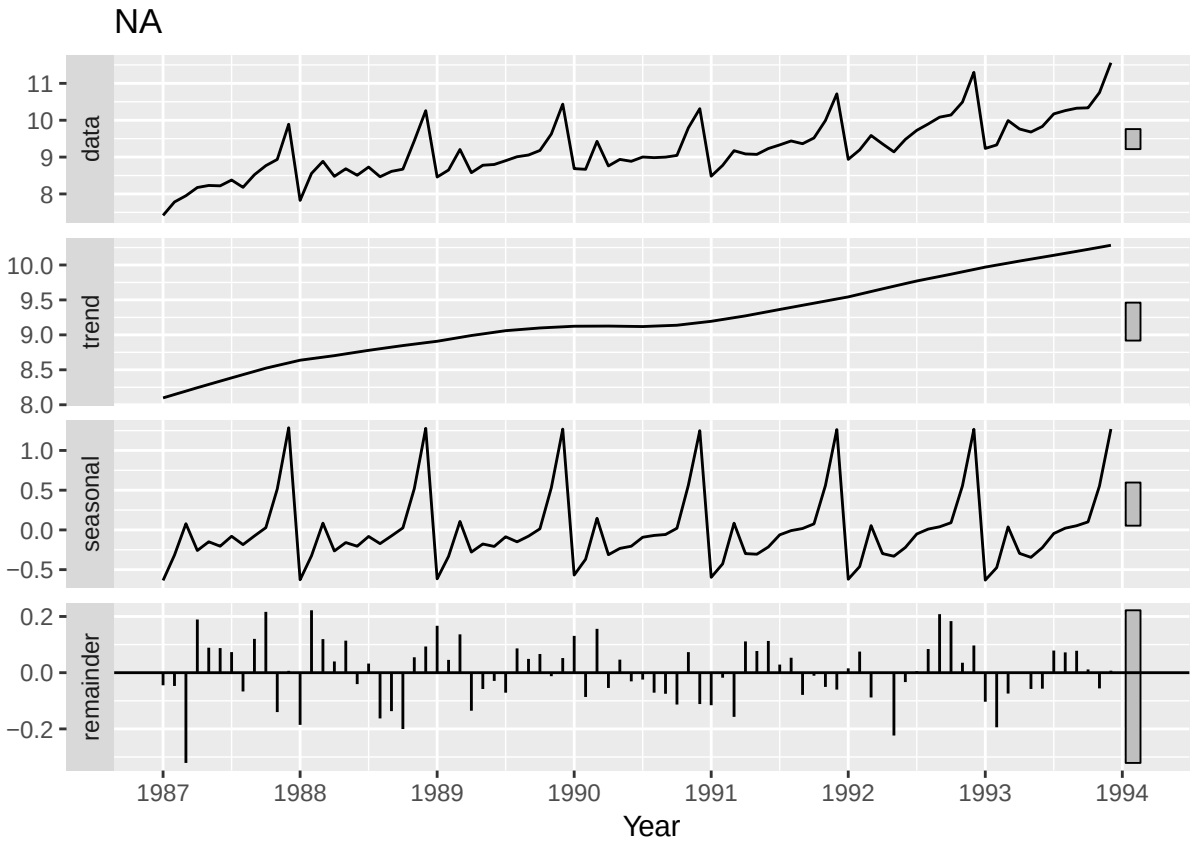
STL

Ventajas:

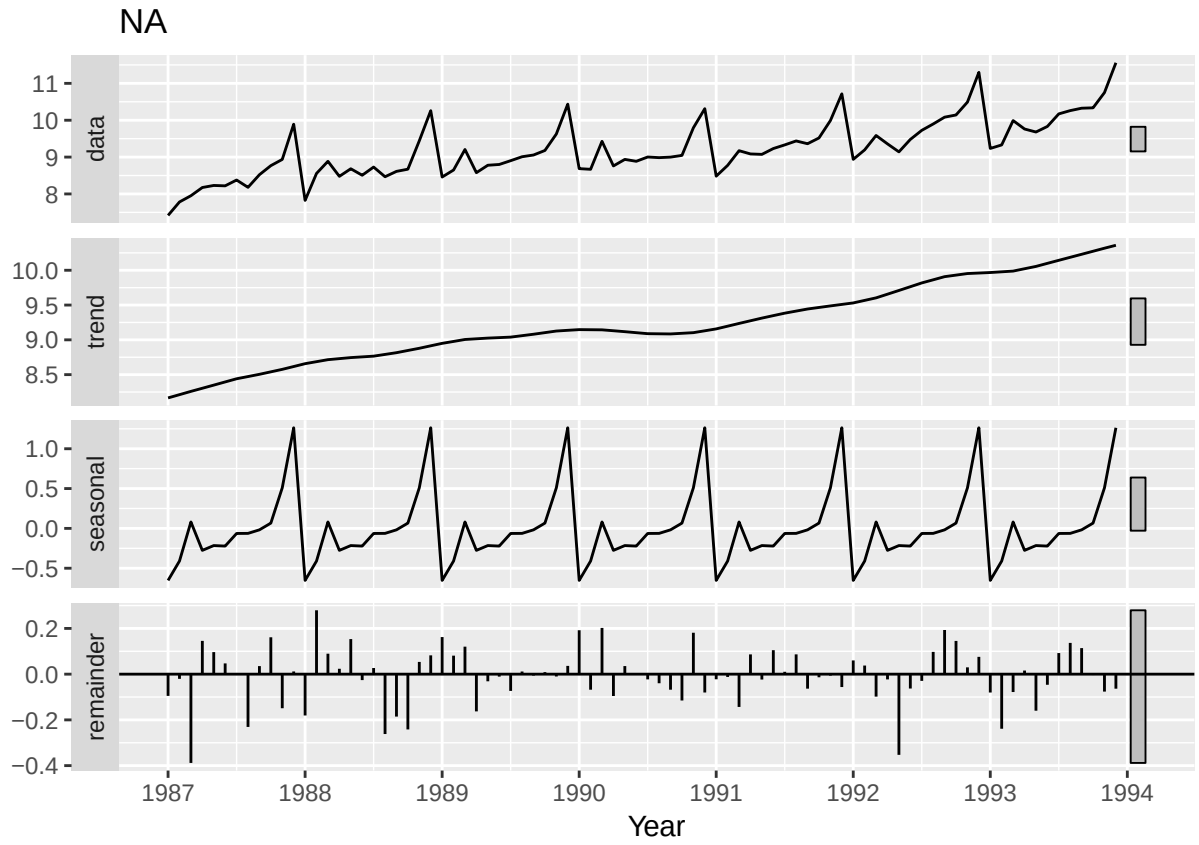
- Puede ajustar datos de todo tipo de frecuencias
- LKa componente estacional puede cambiar con el tiempo y puede ser controlada por el usuario
- Hace una estimación *loess* de la tendencia
- Robusta a outliers
- Usa Box-Cox transformaciones para descomponer

Desventajas: * Solo para series additivas (transformacion logaritmica para las multiplicativas)

```
fit <- stl(souvenirtimeserieslog,s.window=7)
autoplot(fit)+xlab("Year")
```




```
fit2 <- stl(souvenirtimeserieslog,s.window="periodic",t.window=15,robust=T)
autoplot(fit2)+xlab("Year")
```



s.window controla la variación de la componente estacionaria. *t.window* controla como de recta es la tendencia.

Ajuste estacional de los datos

Se trata de quitar la componente estacional de los datos para observar mejor la tendencia. La forma en la que lo hacemos dependerá de si la serie es multiplicativa o aditiva:

- Aditivas:

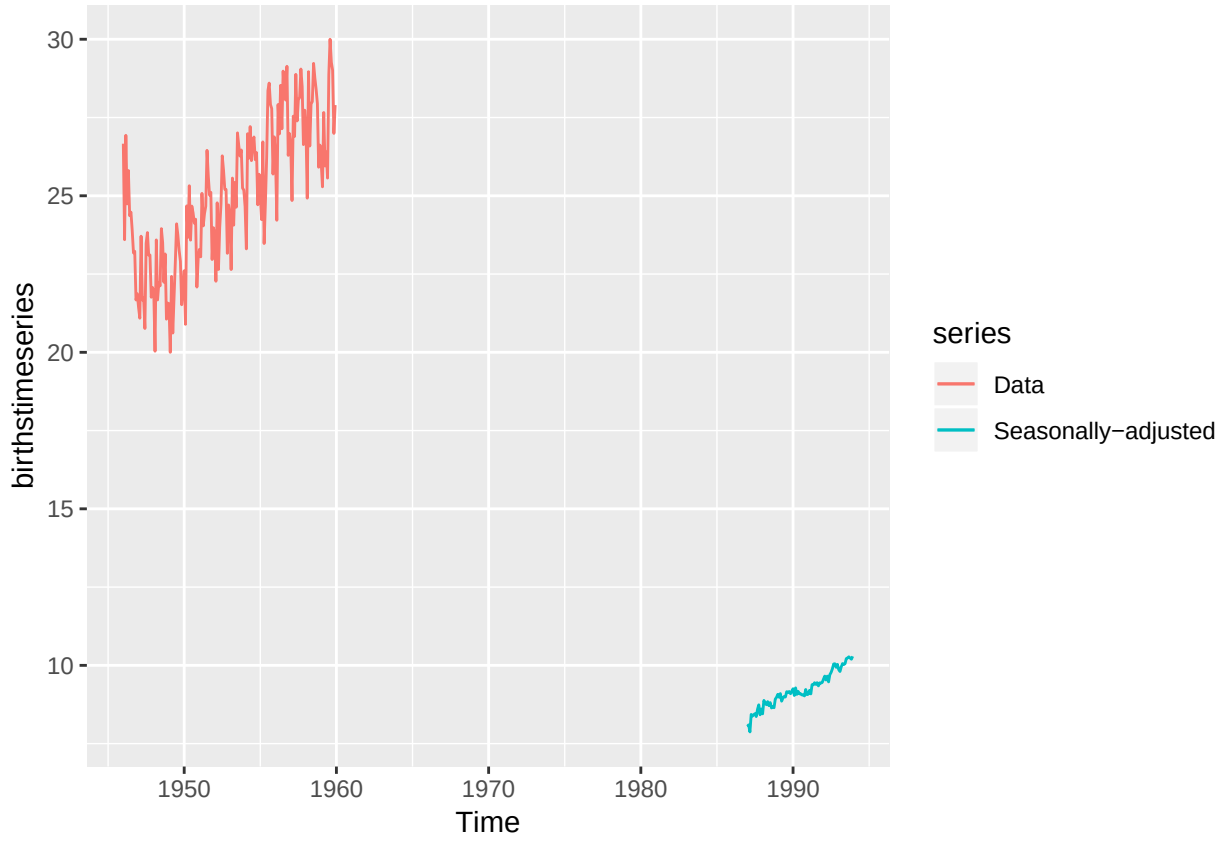
$$Y_t - S_t = T_t + \epsilon_t$$

- Multiplicativas

$$Y_t / S_t = T_t \epsilon_t$$

Al quitar la componente de estacionalidad nos quedamos sólo con la tendencia y el ruido

```
autoplot(birthstimeseries, series="Data") +
autolayer(seasadj(fit), series="Seasonally-adjusted")
```

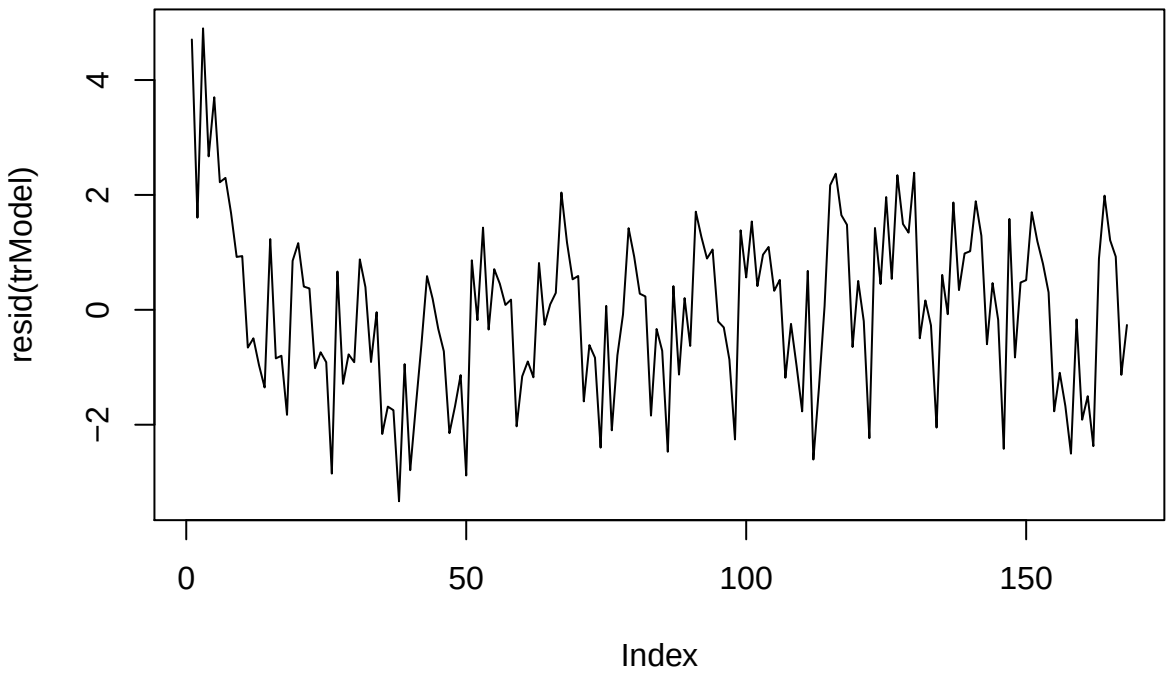


Para ajustar estacionalmente los datos utilizamos valores pasados de esta componente para ajustar estacionalmente un valor actual. Sin embargo, dado que no son datos *suavizados* no es una buena forma de buscar puntos de cambio de tendencia.

Si recordais es una práctica habitual en estadísticas como el paro, donde se intenta ver la tendencia independientemente de la estación del año, que se sabe que produce muchos altos y bajos. Sin embargo en Australia el Bureau of Statistics (ABS) decidió empezar a repotar los datos sin este ajuste debido a un grave error que cometieron. Esto se debió a que en agosto se suele realizar una encuesta a aquellas personas con trabajo. En el 2014 una gran cantidad de gente dijo que estaban desempleados para no tener que responderla. ABS desde entonces utiliza un nuevo método de prediccion sin ajuste estacional.

Ajuste de tendencia de los datos (de-trend)

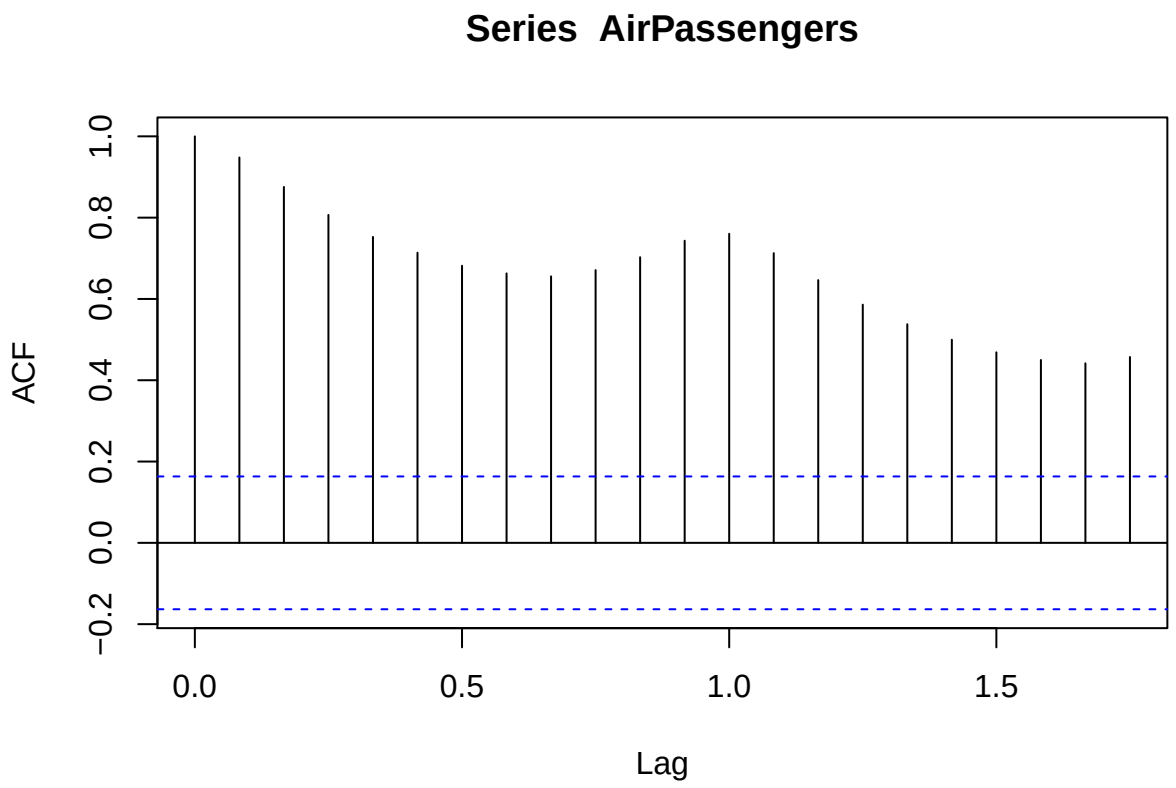
```
trModel=lm(birthstimeseries~seq(1,length(birthstimeseries)))
plot(resid(trModel), type="l")
```



Autocorrelación

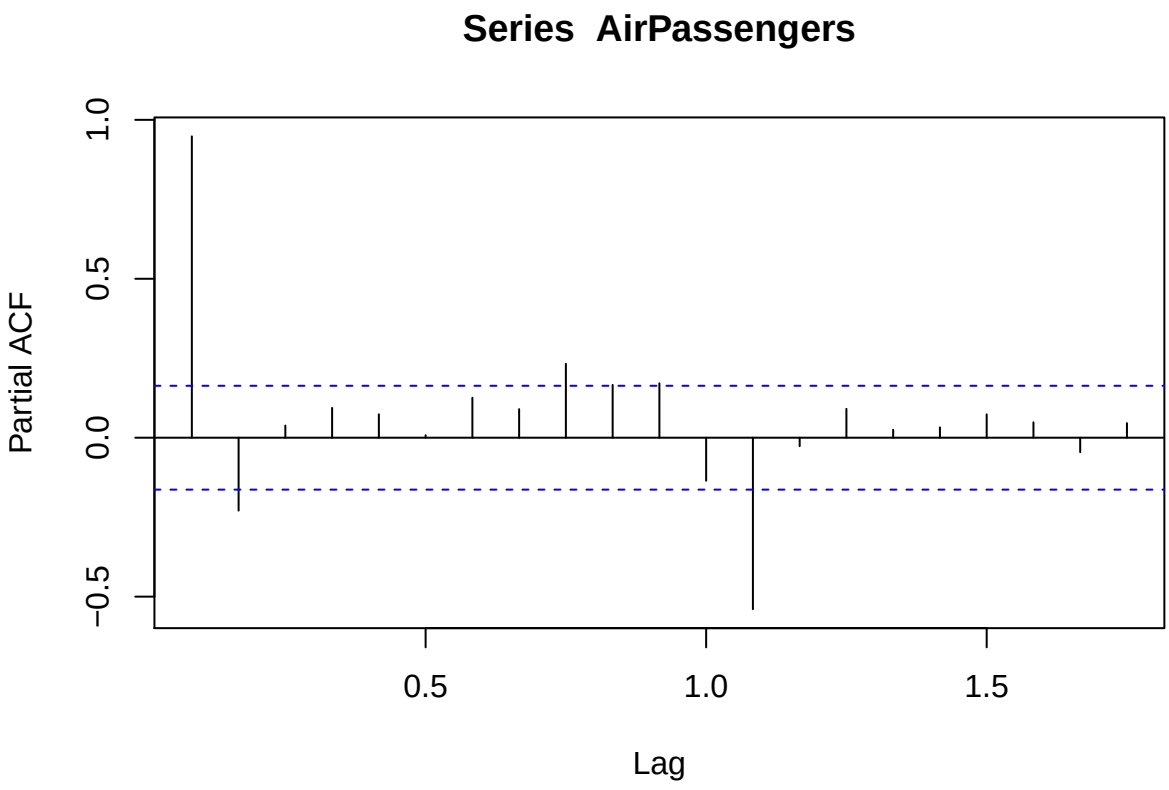
La autocorrelación es la correlación de una variable con "lags" o retardos de si misma.Si existe autocorrelacion quiere decir que en esa variable unos puntos dependen de los anteriores y podemos predecir lo que sucederá en el futuro. Una serie temporal estacionaria tendrá una autocorrelación cayendo a 0 rápidamente, mientras que ese efecto será más gradual en una serie no estacionaria.

```
acfRes <- acf(AirPassengers) # autocorrelation
```

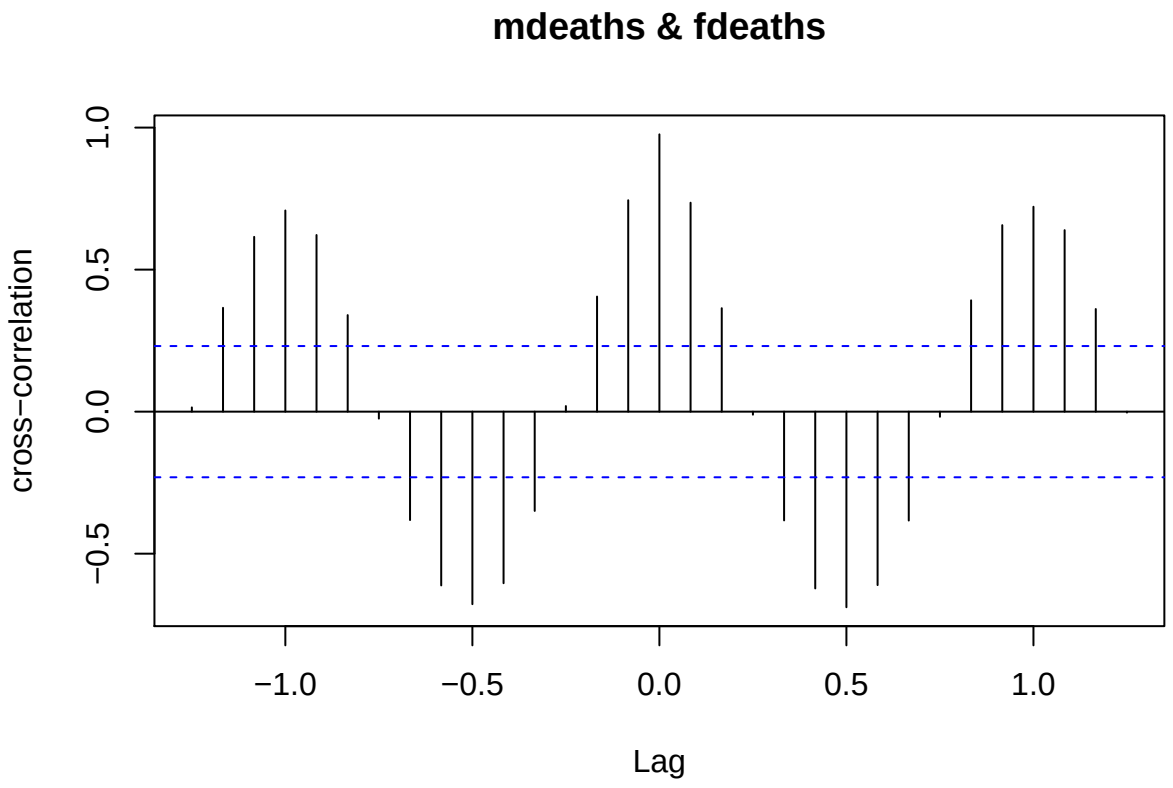


Partial Autocorrelation es la correlacion de la serie temporal con retardos de si misma, una vez hemos quitado la relacion lineal entre puntos.

```
pacfRes <- pacf(AirPassengers)  # partial autocorrelation
```




```
ccfRes <- ccf(mdeaths, fdeaths, ylab = "cross-correlation") # computes cross correlation between 2 time
```

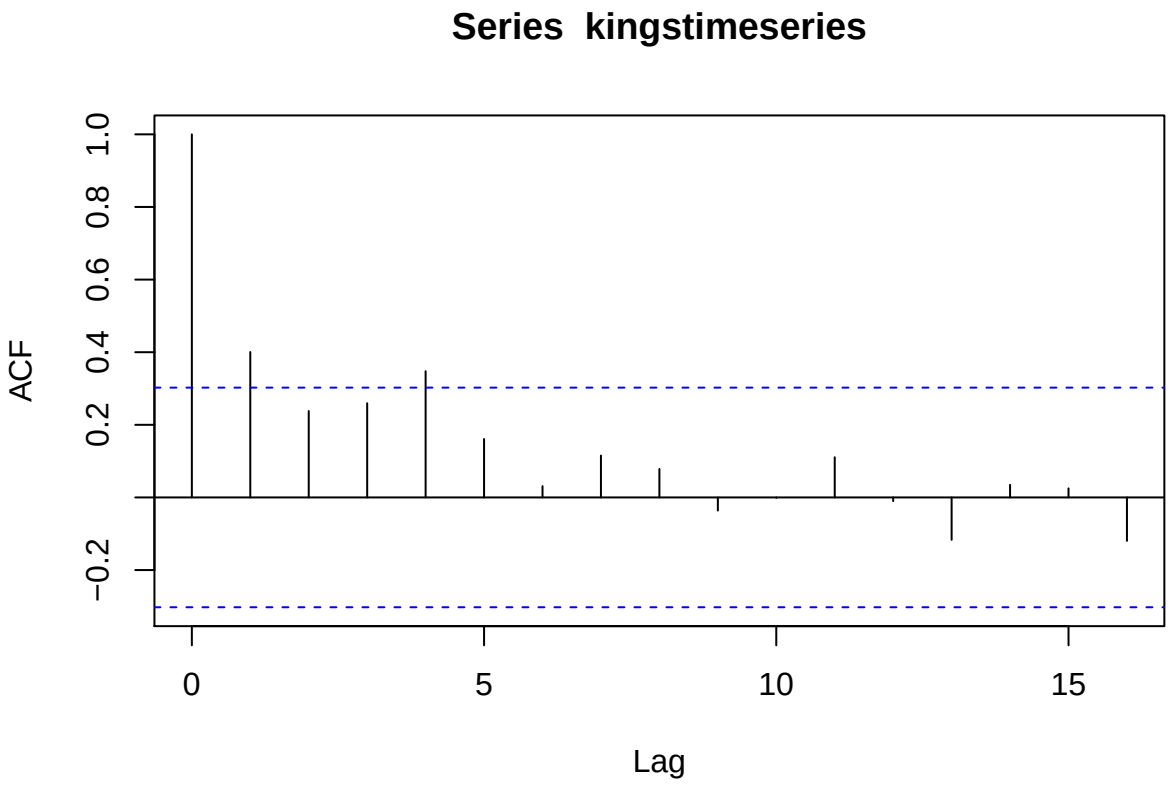



```
head(ccfRes[[1]])
```

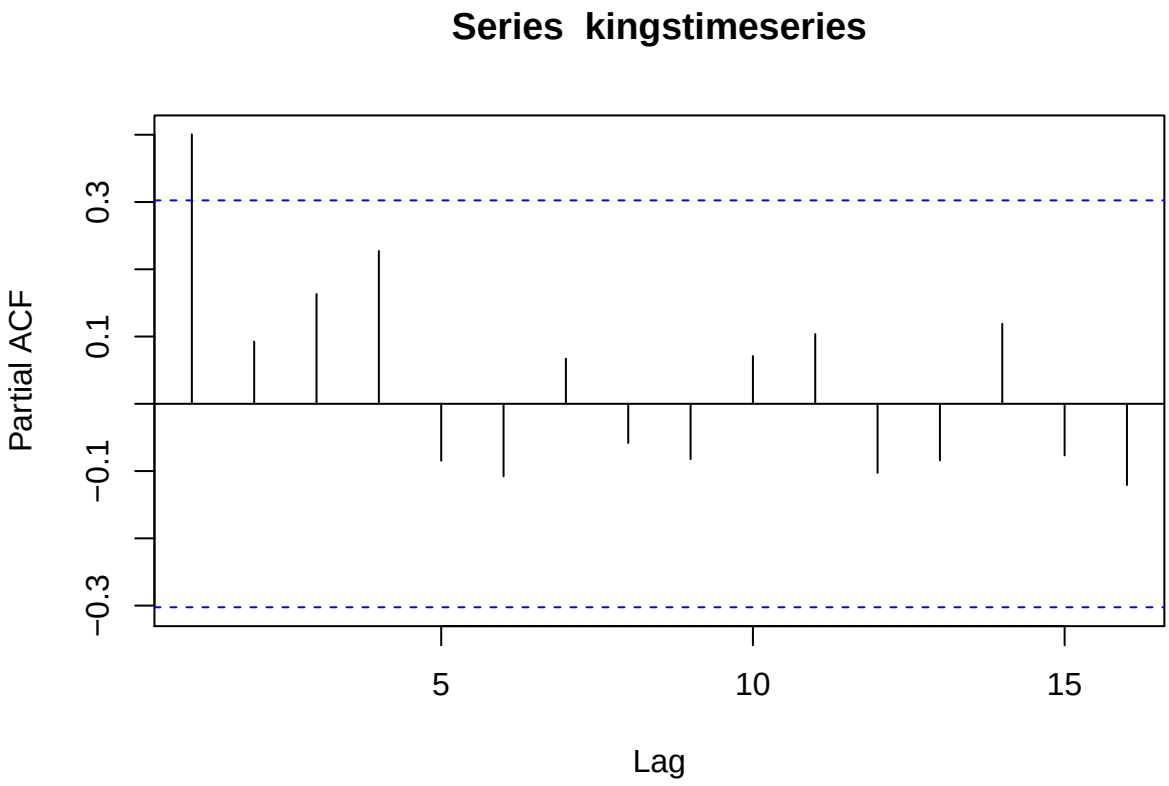
```
## [1] 0.01505498 0.36562603 0.61542712 0.70820629 0.62189580 0.34000545
```

Vamos a verlo con nuestros datasets:

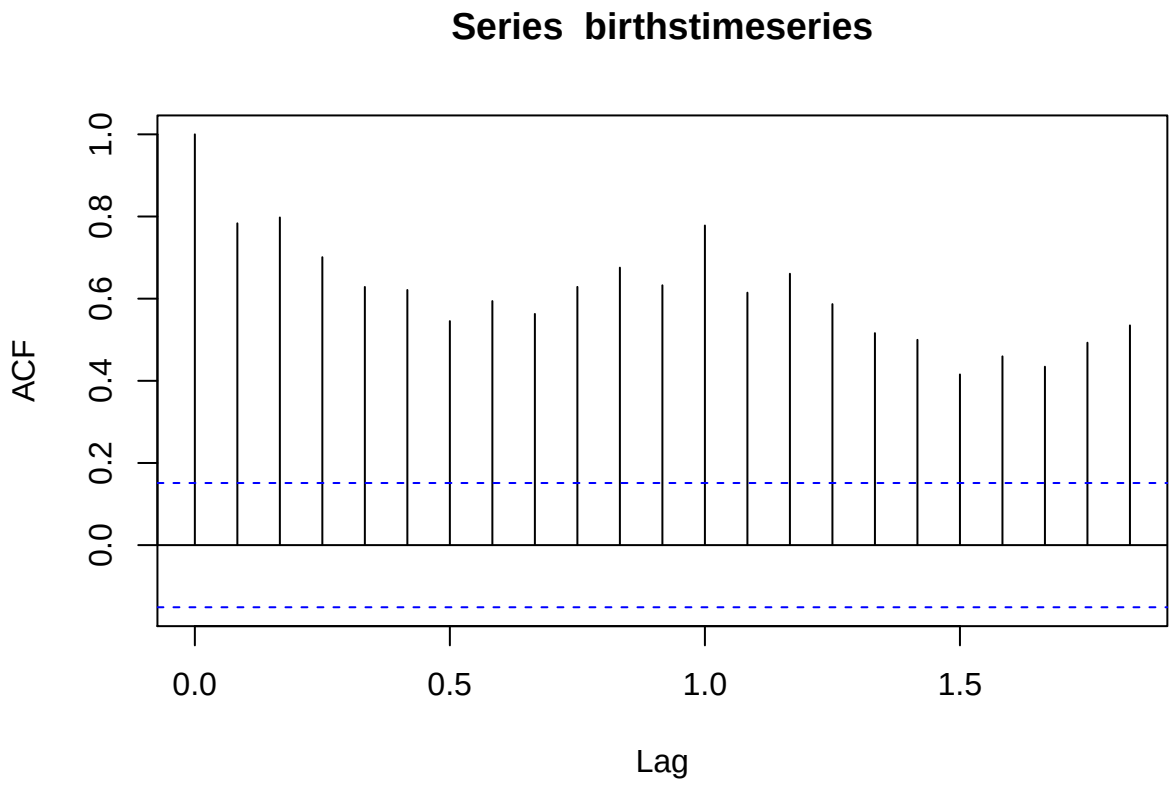
```
acfkings<- acf(kingstimeseries) # autocorrelation
```



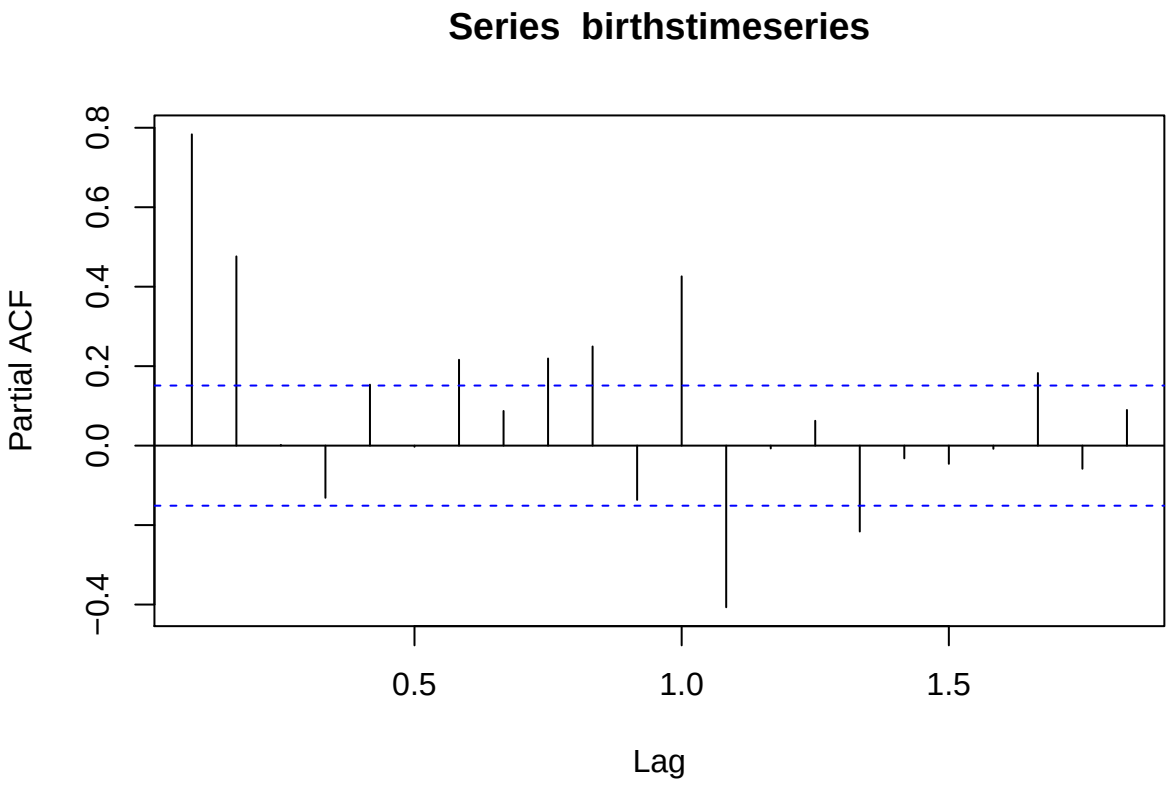

```
pacfRes <- pacf(kingstimeseries)
```



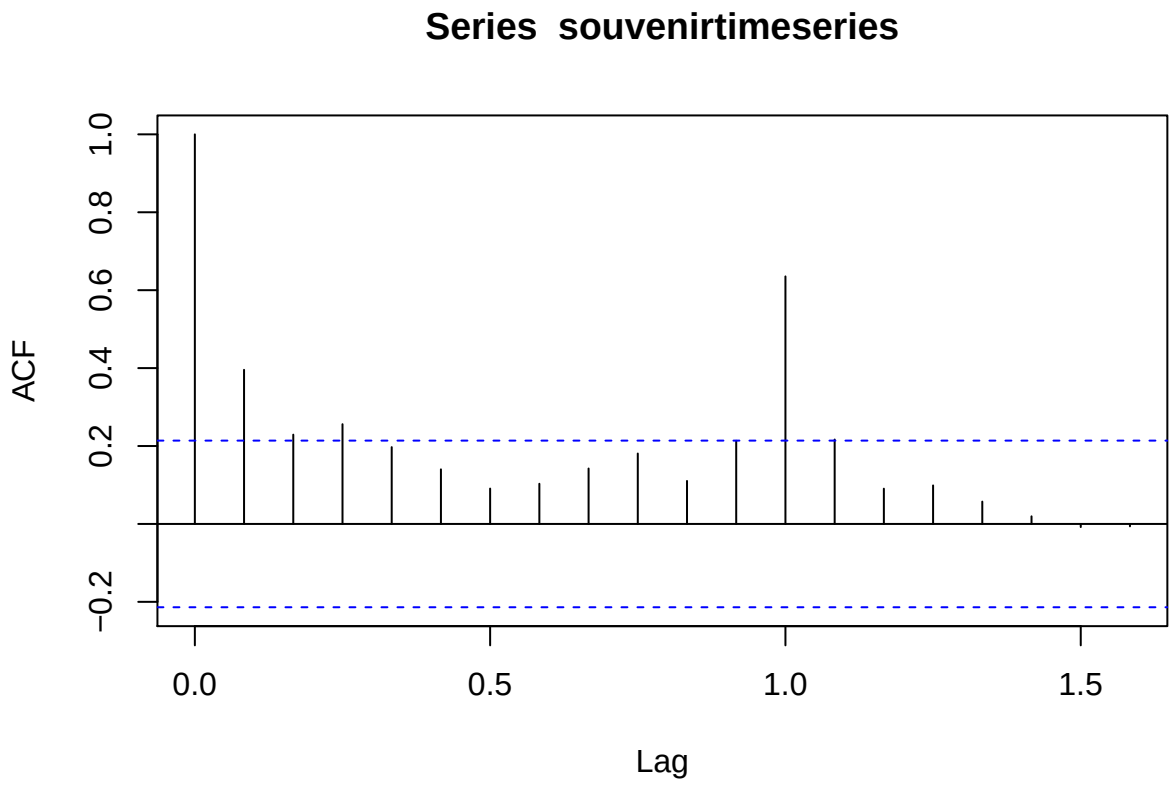

```
acfbirth<- acf(birthstimeseries) # autocorrelation
```



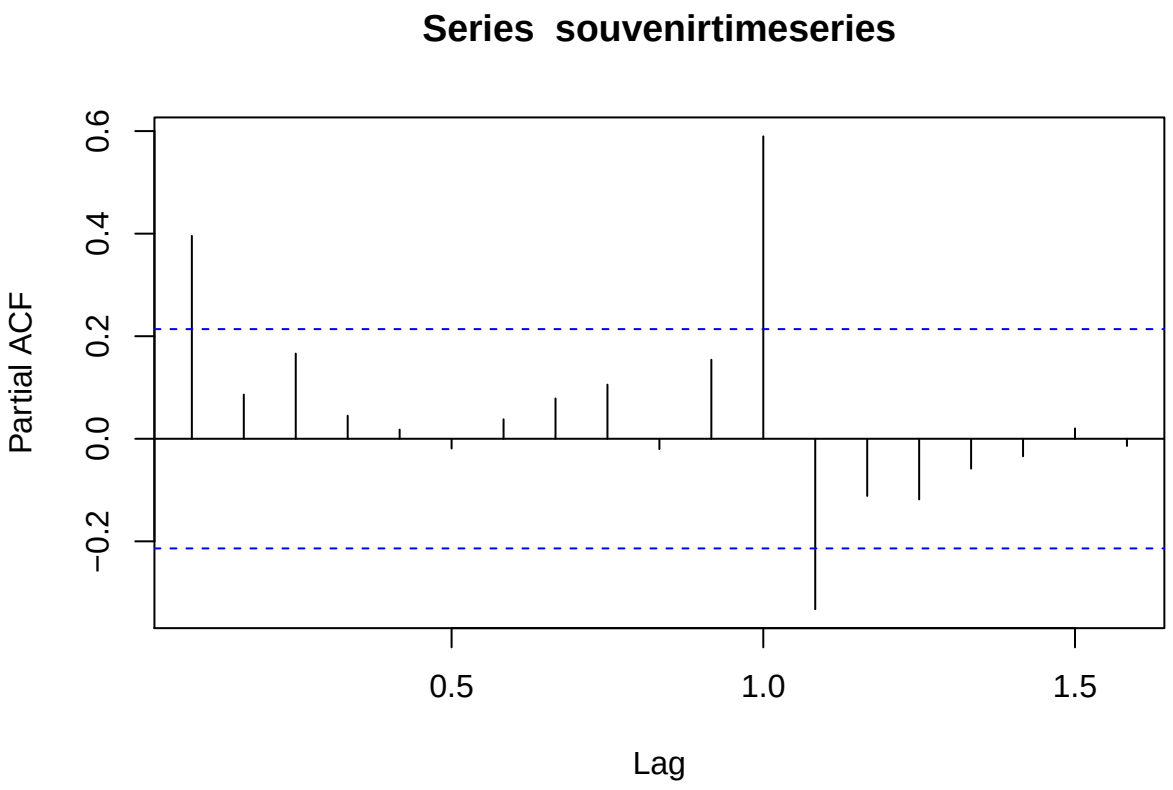

```
pacfbirth <- pacf(birthstimeseries)
```



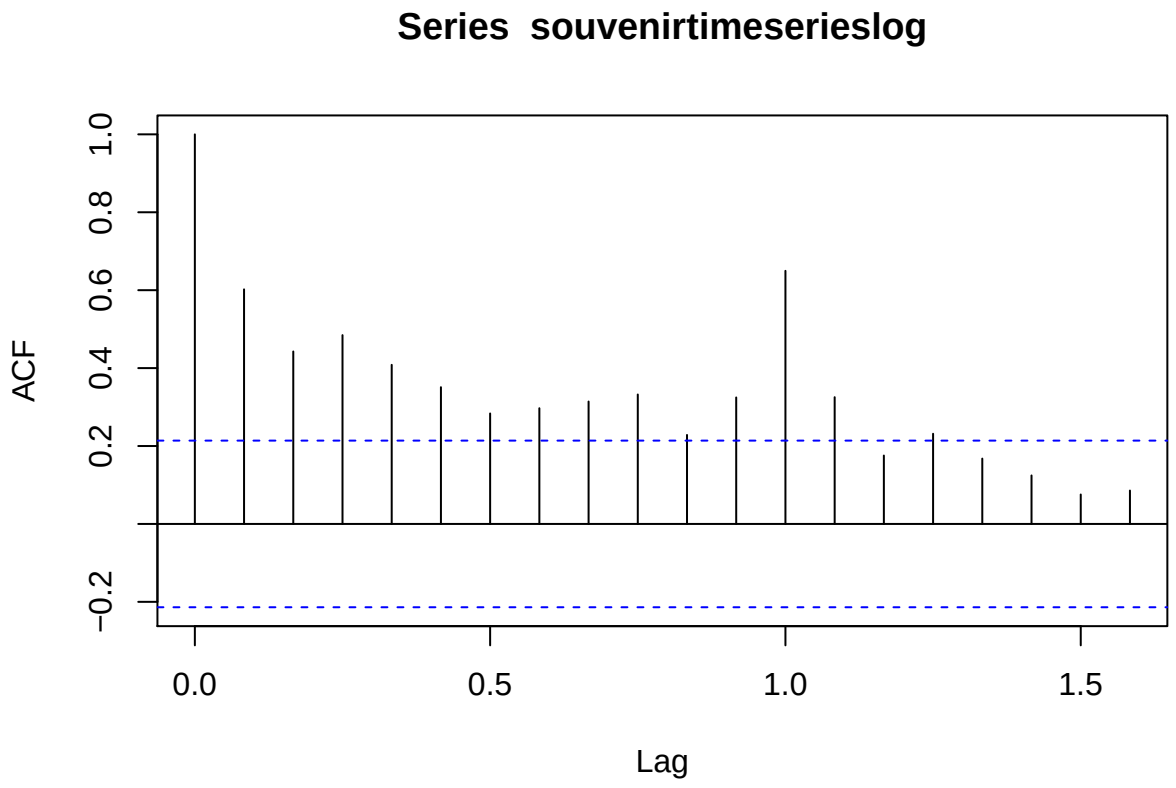

```
acfsouvenir<- acf(souvenirtimeseries) # autocorrelation
```



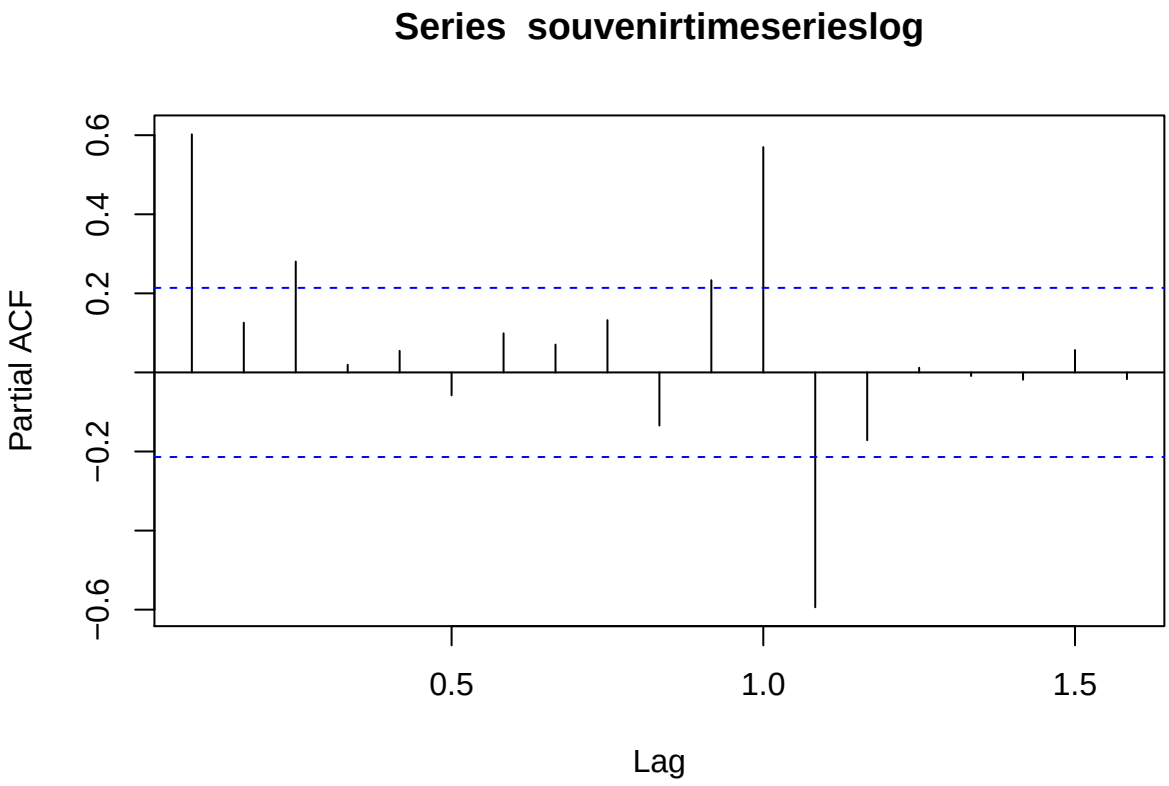

```
pacfsouvenir <- pacf(souvenirtimeseries)
```




```
acfsouvenir<- acf(souvenirtimeserieslog) # autocorrelation
```




```
pacfsouvenir <- pacf(souvenirtimeserieslog)
```



Referencias

https://a-little-book-of-r-for-time-series.readthedocs.io http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/EDescrip/tema7.pdf https://robjhyndman.com/teaching/ https://campus.datacamp.com/courses/forecasting-using-r/exploring-and-visualizing-time-series-in-r?ex=1