CONTENIDO

1. Data Sciencie

1.1. ¿ De qué va esto del Data Science?

La Ciencia de Datos es un campo que intenta extraer ideas e información útil, tanto de datos estructurados como los no estructurados, las cuales pasan por las siguientes etapas:

1. Enmarcar el problema . Hacer las preguntas adecuadas.
   a) ¿ Cuál es el objetivo del análisis o de la empresa?.
   b) ¿ Qué queremos estimar o predecir?
2. Adquirir y preparar los datos.
   a) ¿ Qué recursos tenemos para obtener datos?
   b) ¿ Qué información es relevante?
   c) Limpiar y filtrar los datos para su posterior análisis.
3. Explorar los datos.
   a) Visualizar los datos. Con esto se puede encontrar: tendendicas, correlaciones, patrones entre diferentes valoees. los gráficos son una herramienta muy útil.
   b) Localizar en los gráficos posibles tendencias, correlaciones o patrones.
4. Modelizar y evaluar los datos.
   a) Utilizar algún algoritmo innovador (según el problema) para crear el modelo.
   b) Evaluar el modelo.
5. Comunicar los resultados y/o puesta en producción.
   a) ¿ Qué resultados hemos obtenido?
   b) ¿ Qué hemos aprendido?
   c) ¿ Los resultados tienen sentido?

En cualquier etapa del análisis de datos puede ser posible ir un paso atrás en la etapa, y reemplantear algo que no se haya tenido en consideración.

Normalmente un Científico de Datos o analista, intenta cargar o recolectar datos desde diferentes fuentes externas y que cada una de ellas tiene un formato diferente, por lo cual se lee y se importa de forma diferente en R.
Los datasets que nos encontramos suelen tener las siguientes dificultades para poder analizarlos.

- Faltan valores.
- Están desordenados.
- Se encuentran en formatos que no son posibles de leer.
- Encontramos valores que se salen de lo esperado.

Antes de empezar a analizar los datos, tenemos que: Hacer una limpieza para poder eliminar cosas que nos sobran e inferir algunas que nos faltan.

1.2. El fichero CSV y la función read.csv

Un archivo csv (valores separados por coma) es un archivo de texto que almacena los datos en forma de columnas, separadas por coma y las filas se distinguen por saltos de línea. Son más rápidos de procesar y pueden ser leídos por una gran cantidad de programas.

CSV con encabezado

```
library(readr)
auto <- read.csv("..//r-course-master/data/tema1/auto-mpg.csv") #Crea un Dataframe
names(auto) #Names indica los nombres de las variables del encabezado

## [1] "No"           "mpg"          "cylinders"    "displacement" "horsepower"
## [6] "weight"       "acceleration" "model_year"   "car_name"

head(auto, 3)

##   No mpg cylinders displacement horsepower weight acceleration model_year
## 1  1  28         4          140         90   2264         15.5         71
## 2  2  19         3           70         97   2330         13.5         72
## 3  3  36         4          107         75   2205         14.5         82
##              car_name
## 1 chevrolet vega 2300
## 2     mazda rx2 coupe
## 3        honda accord
```

CSV sin encabezado

```
auto_noheader <- read.csv("..//r-course-master/data/tema1/auto-mpg-noheader.csv",
                          header = FALSE,col.names = c("número", "millas", "Cilindrada",
                          "Desplazamiento", "wph", "peso", "aceleración", "año", "modelo")
                          ,stringsAsFactors = FALSE)
names(auto_noheader) #Names indica los nombres de las variables del encabezado

## [1] "número"         "millas"         "Cilindrada"     "Desplazamiento"
## [5] "wph"            "peso"           "aceleración"    "año"
## [9] "modelo"

head(auto_noheader, 3)

##   número millas Cilindrada Desplazamiento wph peso aceleración año
## 1      1     28          4            140  90 2264        15.5  71
## 2      2     19          3             70  97 2330        13.5  72
## 3      3     36          4            107  75 2205        14.5  82
##               modelo
## 1 chevrolet vega 2300
## 2     mazda rx2 coupe
## 3        honda accord
```

CSV online

```
who_internet <- read.csv("https://frogames.es/course-contents/r/intro/tema1/WHO.csv")

## Warning in file(file, "rt"): fallo en InternetOpenUrl: Ño se pudo resolver el nombre
de servidor ni su dirección
## Error in file(file, "rt"): no se puede abrir la conexión
```

- Los strings por defecto se guardan como factores (o variables categóricas).

- Para leer un CSV desde internet se usa la misma instrucción que para leer un csv local
- Para convertir una variable categórica a carácter, se usa la instrucción **as.character()**

1.3. El fichero XML

Aveces necesitamos extraer información de una página web (scraping web). La mayoría de web proporcionan información en 2 formatos muy estandarizados:

1. XML
2. JSON

```
library(XML)
#Creamos un apuntador que nos lozalice el documento.
doc <- xmlParse("..//r-course-master/data/tema1/cd_catalog.xml") #CML InternalDocument
rootnode <- xmlRoot(doc) #nos coloca en el Nodo Raíz del fichero
rootnode[2] #Primer elemento del nodo raíz.

## $CD
## <CD>
##   <TITLE>Hide your heart</TITLE>
##   <ARTIST>Bonnie Tyler</ARTIST>
##   <COUNTRY>UK</COUNTRY>
##   <COMPANY>CBS Records</COMPANY>
##   <PRICE>9.90</PRICE>
##   <YEAR>1988</YEAR>
## </CD>
##
## attr(,"class")
## [1] "XMLInternalNodeList" "XMLNodeList"

cds_data <- xmlSApply(rootnode, function(x) xmlSApply(x, xmlValue) )
#Devuelve una matriz de los hijos raiz con sus respectivos valores
cds.catalog <- data.frame(t(cds_data), row.names = NULL )
#Convierte en DataFrame y lo transpone
head(cds.catalog,2) #Me muuestra las 2 primeras filas

##              TITLE       ARTIST COUNTRY     COMPANY PRICE YEAR
## 1 Empire Burlesque    Bob Dylan     USA    Columbia 10.90 1985
## 2  Hide your heart Bonnie Tyler      UK CBS Records  9.90 1988

cds.catalog[1:5,1:3] #Me muestra las primero 5 filas con las 3 primeras columnas

##                 TITLE          ARTIST COUNTRY
## 1    Empire Burlesque       Bob Dylan     USA
## 2     Hide your heart    Bonnie Tyler      UK
## 3       Greatest Hits    Dolly Parton     USA
## 4 Still got the blues      Gary Moore      UK
## 5               Eros Eros Ramazzotti      EU
```

En XML no obtenemos todos los datos, sino que hay que extraer los datos del XML

Los niveles de profundidad que puede llegar a tener un XML pueden ser realmente complejos e incluso difíciles de extraer Conocer un poco de sintaxis Xpath es útil para acceder a determinadas etiquetas de XML. Básicamente las funciones que se utilizará son:

- xpathSapply
- getNodeSet()

Leyendo tablas incrustradas en un HTML

HTML es ana forma especializada de una forma muy concreta de XML. R nos da opciones para trabajar con tablas HTML .

Cargar todas las tablas que están en el HTML y luego sacar la tabla que nos interesa.

```
library(XML)
population_url <- "..//r-course-master/data/tema1/WorldPopulation-wiki.htm"
#Con una variante podemos extraer todas las tablas de HTML, con la función readHTMLTable
tables <- readHTMLTable(population_url)
#Se pone de esta manera ya que es una lista de lista
#(6 es el número de la lista que quiero acceder)
most_populated <- tables[[6]]
head(most_populated,3)

##   Rank Country / Territory    Population              Date
## 1    1       China[note 4] 1,385,310,000 September 9, 2017
## 2    2               India 1,321,010,000 September 9, 2017
## 3    3       United States   325,732,000 September 9, 2017
##   Approx.Â % of world\npopulation Source
## 1                           18.3%   [91]
## 2                           17.5%   [92]
## 3                           4.31%   [93]
```

Pero también podemos extraer una de las tablas si se conoce que número de tabla queremos extraer.

```
#Es una forma de no cargar todas las tablas que tiene una página web
Tabla_6 <- readHTMLTable(population_url,wich =6)
```

1.4. El fichero JSON

La mayoría de APIs rest que existen por la web se fundamentan en web service, los cuales devuelven datos en un formato llamado JSON o JavaScript Object Notation, ya que en muchos modos es más simple y mucho más eficiente que el *XML* o *CSV*.

- Se sigue indicando con etiquetas y varios niveles de profundidad el acceso a cada uno de los datos.
- Cada uno de los objetos está englobado entre llaves, esta compuesta por etiqueta valor.

En este caso usaremos dos ficheros de datos de tipo JSON

```
library(jsonlite)
#fromJSON es la encarga de cargar directamente los datos de un fichero de esta extensión.
dat.1 <- fromJSON("..//r-course-master/data/tema1/students.json")
dat.2 <- fromJSON("..//r-course-master/data/tema1/student-courses.json")
dat.1[c(3,5,7),]

##     id            Name               Email     Major year
## 3 1046   Jordan Keller       Cum@gmail.com Chemistry 2014
## 5 1010   Deanna Ortega Dea.Ortega@velit.ca Chemistry 2016
## 7 1051 Yoko Washington    yoko@yahoo.co.uk Marketing 2017

dat.2[,c(2,4)]

##                 Name      Major
## 1 Frances H. Morton Statistics
## 2        Tim Norton Statistics
```

Una de las páginas web donde se obtiene cargar datos de internet en formato JSON puede ser, por ejemplo, Yahoo Finance

1.5. Ficheros de ancho fijo

Estos ficheros son un archivo de texto de ancho fijo, los datos están contenidos en columnas que tienen un ancho fijo (lo que significa que cada columna contiene un cierto número de posiciones de caracteres). Los datos que no llenen el ancho de la columna se rellenan con caracteres de relleno, que suelen ser espacios o ceros.

Sin encabezado

```
students_data <- read.fwf("..//r-course-master/data/tema1/student-fwf.txt",
                          widths=c(4, 15, 20, 15, 4),
                          col.names = c("id", "nombre", "email",
                                        "carrera", "año")
                          )
#widthes -> Especificas las anchuras
#col.names -> Añade encabezado
#fwf -> Formateado con el Ancho Fijo
head(students_data, 3)

##     id          nombre                email         carrera  año
## 1 1044 Fran Morton     fran.m@quama.edu     Statistics      2015
## 2 1035 Kato Mullins    tempor@giat.com      Marketing       2014
## 3 1046 Jordan Keller   Cum@gmail.com        Chemistry       2014
```

Con encabezado

En estos ficheros, el nombre de las columnas no respeta la anchura fija, por eso es necesario incorporar el separador que separa SOLAMENTE las cabeceras del fichero.

```
students_data_header <- read.fwf("..//r-course-master/data/tema1/student-fwf-header.txt",
                                 widths = c(4,15,20,15,4),
                                 header = TRUE, sep = "\t",
                                 skip = 2)
#sep = "\t" -> Es el separador de las cabeceras.
#skip -> Separa el no. de lineas (las quita)
head(students_data_header,3)

##     ID            Name                Email           Major Year
## 1 1044 Fran Morton     fran.m@quama.edu     Statistics      2015
## 2 1035 Kato Mullins    tempor@giat.com      Marketing       2014
## 3 1046 Jordan Keller   Cum@gmail.com        Chemistry       2014
```

Para eliminar alguna columna, hacemos lo siguiente:

```
students_data_no_email <- read.fwf("..//r-course-master/data/tema1/student-fwf.txt",
                          widths=c(4, 15, -20, 15, 4),
                          col.names = c("id", "nombre",
                                        "carrera", "año")
) #Nota: Quite la columna 3, así que por eso se le pone -20
head(students_data_no_email,3)
```

```
##     id          nombre         carrera  año
## 1 1044 Fran Morton     Statistics      2015
## 2 1035 Kato Mullins    Marketing       2014
## 3 1046 Jordan Keller   Chemistry       2014
```

1.6. Ficheros Rdata y rds

Los ficheros en los que R almacena los distintos objetos creados en el espacio de trabajo o workspace tienen extensión .RData.

Creación de fichero Rdata y rds

```
clientes <- c("Carlos", "Chucho", "Yurisad")
fechas <- as.Date(c("2021-11-8","2021-11-19","2021-11-14"))
#as.Date() ->Recordemos que a las fechas se les puede sumar días, meses, etc. por eso se
#convierte a tipo Date
pago <- c(879,658,712)
pedidos <- data.frame(clientes,fechas,pago) #data.frame()-> Nos crea un DataFrame
head(pedidos) #Nos muestra el DataFrame creado

##   clientes     fechas pago
## 1   Carlos 2021-11-08  879
## 2   Chucho 2021-11-19  658
## 3  Yurisad 2021-11-14  712

save(pedidos,file = "../Pedidos.Rdata")
#Guarda mi DataFrame con el nombre de pedidos y con extensión Rdata
saveRDS(pedidos,file = "../Pedidos.rds")
#Guarda mi DataFrame con el nombre de pedidos y con extensión rds
remove(pedidos)
#remove() -> Limpiar la memoria RAM (cuando dejemos de trabajar con el DataFrame)
```

Cargar fichero Rdata y rds

- Para cargar un fichero rds, primero hay que crear la variable donde se guardará, esta es una de la diferencia con el fichero Rdata.
- La función *saveRDS()* es una función que solamente es capaz de guardar un objeto, es decir, no guarda el nombre de la variable donde staba dicho objeto.

```
load("../Pedidos.Rdata") #Cargando un fichero Rdata
pedidos_rds <- readRDS("../Pedidos.rds")
#Cargando un fichero rds, guardando el DataFrame en la variable pedidos_rds
head(pedidos_rds)

##   clientes     fechas pago
## 1   Carlos 2021-11-08  879
## 2   Chucho 2021-11-19  658
## 3  Yurisad 2021-11-14  712
```

Existe la opción de guardar todos los objetos de la sesión. Esto sirve para guardar toda la sesión y luego abrirla en otro ordenador. Para esto utilizaremos la función *save.image()*:

```
save.image(file = "../alldata.Rdata")
```

```
primos <- c(2,3,5,7,11,13)
impares <- c(1,3,5,7,9,11,13)
save(list = c("primos","impares"), file = "../primos_e_impares.Rdata")
```

1.7. Manipulación de datos

1.7.1. Eliminar datos sin valor con nao.mit

Cuando nos falta información, una de las opciones es, eliminar estos casos directamente del conjunto de datos cargados.

```
library(readr)
MISSING_DATA <- read.csv("../r-course-master/data/tema1/missing-data.csv", na.strings = "")
#na.string="" -> Nos sirve para rellenar los espaios en blanco con NA
head(MISSING_DATA,6)

##   Income Phone_type   Car_type
## 1  89800    Android     Luxury
## 2  47500    Android Non-Luxury
## 3  45000     iPhone     Luxury
## 4  44700       <NA>     Luxury
## 5  59500     iPhone     Luxury
## 6     NA    Android Non-Luxury
```

Para obtener un DataFrame que no tenga datos faltantes, se puede hacer uso de la función *na.omit*

```
MISSING_DATA_cleaned <- na.omit(MISSING_DATA)
head(MISSING_DATA_cleaned)

##   Income Phone_type   Car_type
## 1  89800    Android     Luxury
## 2  47500    Android Non-Luxury
## 3  45000     iPhone     Luxury
## 5  59500     iPhone     Luxury
## 7  63300     iPhone Non-Luxury
## 8  52900    Android     Luxury
```

La función *na.omit()*, es una función que internamente utiliza otra función llamada *is.na()*, esta última función nos permite averiguar si alguno de los argumentos que hemos pasado es de tipo NA. Tendremos los siguiente con la función *is.na()*:

- Cuando se aplica a un valor simple, nos devuelve un Booleano.
- Cuando se aplica a una colección, nos devuelve un vector de Booleanos.

```
is.na(MISSING_DATA[4,2]) #Aplicado a un valor simple

## [1] TRUE

is.na(MISSING_DATA$Income) #Aplicado a una colección.

##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [13]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE
```

Para eliminar sólo los casos donde el NA está presente en una de las variables, hacemos lo siguiente:

```
#Limpiando NA solamente de la variable Income
datos_miss <-MISSING_DATA[ !is.na(MISSING_DATA$Income), ]
```

La función *complete.cases()* , es una función que dado un DataFrame o una tabla como argumento, devuelve un vector de Booleanos indicando TRUE para las filas que no tienen casos faltantes

```
complete.cases(MISSING_DATA)

##  [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13] FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE
```

Otra forma de quedarnos con los datos que solamente tengan datos completos, seria:

```
MISSING_DATA_cleaned_2 <- MISSING_DATA[complete.cases(MISSING_DATA), ]
head(MISSING_DATA_cleaned_2)

##   Income Phone_type   Car_type
## 1  89800    Android     Luxury
## 2  47500    Android Non-Luxury
## 3  45000     iPhone     Luxury
## 5  59500     iPhone     Luxury
## 7  63300     iPhone Non-Luxury
## 8  52900    Android     Luxury
```

Avece encontraras tablas en donde traen Datos Atípicos, es decir, al realizar una encuesta puede ser el caso que la persona encuestada no haya respondido esa pregunta con la lógica que queremos. Por ejemplo, poner que la edad máxima que pueda llegar una persona sea 200 años. Con esto lo que debemos hacer es elimina esos datos, ya que al momento de hacer alguna predicción o el promedio de una variable, nos generara conclusiones incorrectas.

```
MISSING_DATA$Income[MISSING_DATA$Income == 0] <- NA  #Convertir los ceros de ingresos en NA
```

Cuando utilizamos medidas de centralización y dispersión (por ejemplo, promedio, desviación estándar), y tenemos valores con NA

```
# na.rm =TRUE -> Nos sirve para calcular la media ignorando los NA
mean(MISSING_DATA$Income, na.rm = TRUE)

## [1] 65763.64
```

```
# na.rm =TRUE -> Nos sirve para calcular la desviación típica ignorando los NA
sd(MISSING_DATA$Income, na.rm = TRUE )

## [1] 26715.87
```

1.7.2. Reemplazar valores

En general, cuando se descartan los casos que tienen valores faltantes, se suele perder información que es útil, para esto, podemos reemplazar los valores que faltan con algo que nos pueda ser de utilidad. Una de las técnicas sería:

Reemplazar los valores con la media

```
#Creo una cuarta columan con el nombre Income.mean en el DataFrame MISSING_DATA
MISSING_DATA$Income.mean <- ifelse(is.na(MISSING_DATA$Income),
                                   mean(MISSING_DATA$Income, na.rm = TRUE),
                                   MISSING_DATA$Income)
head(MISSING_DATA, 6)

##   Income Phone_type   Car_type Income.mean
## 1  89800    Android     Luxury    89800.00
## 2  47500    Android Non-Luxury    47500.00
## 3  45000     iPhone     Luxury    45000.00
## 4  44700       <NA>     Luxury    44700.00
## 5  59500     iPhone     Luxury    59500.00
## 6     NA    Android Non-Luxury    65763.64
```

Reemplazar los valores con Muestreo Aleatorio Simple con reemplazo

Pero aquí hay dos casos que vale la pena discutir, que, no se puede calcular el promedio si la variables es categórica y tiene valores faltantes. Para esto, una de las técnicas en el mundo del Big Data es:

- Computar valores aleatorio, extraídos desde una muestra aleatoria de los datos que no faltan, es decir, se puede haer un muestreo aleatorio (sacar algunos aleatoriamente) y decidir que el valor que falta es el de la extracción.

E siguiente código sirve tanto para variables númericoas como categóricas.

```
MISSING_DATA <- read.csv("../r-course-master/data/tema1/missing-data.csv", na.strings = "")
#Reemplazo los valores que tengan 0 con NA
MISSING_DATA$Income[MISSING_DATA$Income == 0] <- NA

rand.impute <- function(x){ #x es un vector de datos que puede contener NA
 # missing_al es un vector del mismo tamaño que x, contiene un vector de valores T/F
  #dependiendo del NA de x
  missing_al <- is.na(x)
  #n.missing contiene cuantos valores son NA
  n.missing_al <- sum(missing_al)
  #x.obs son los valores que tienen dato diferente de NA en x.
  x.obs <- x[!missing_al]
  #por defecto devolveré lo mismo que había entrado por parámetro
  imputed <- x
  #En los valores que faltaba, los reemplazamos por una muestra de los que si conocemos
  imputed[missing_al] <- sample(x.obs , n.missing_al , replace = TRUE)
  return(imputed)
}

random.impute.data.frame <- function(Data_frame_par, cols){
  names <- names(Data_frame_par)
  #Me quedó con los nombres que llega como parametro
  for (col in cols){
    #col in cols -> Por cada columna dentro del array dado por parámetro (cols)
    name <- paste(names[col], "imputed", sep = ".")
    #name concatena names en la posición col, con el nombre col.imputed
```

```
    Data_frame_par[name]=rand.impute(Data_frame_par[ , col])
    #dataframe[name] -> Asigna el dataframe dado por parámetro en la columna name
    #(col.imputed) el valor de mi función aleatoria
  }
  Data_frame_par
}
#Le doy  mi función los parametros: El DataFrame que quiero que le aplique el Muestreo
#Aleatorio Simple con reemplazo y las columnaas (1 y 2) a las que se lo aplicará
random.impute.data.frame(MISSING_DATA, c(1,2))
```

```
##    Income Phone_type   Car_type Income.imputed Phone_type.imputed
## 1   89800    Android     Luxury          89800            Android
## 2   47500    Android Non-Luxury          47500            Android
## 3   45000     iPhone     Luxury          45000             iPhone
## 4   44700       <NA>     Luxury          44700             iPhone
## 5   59500     iPhone     Luxury          59500             iPhone
## 6      NA    Android Non-Luxury         145100            Android
## 7   63300     iPhone Non-Luxury          63300             iPhone
## 8   52900    Android     Luxury          52900            Android
## 9   78200    Android     Luxury          78200            Android
## 10 145100     iPhone     Luxury         145100             iPhone
## 11  88600     iPhone Non-Luxury          88600             iPhone
## 12  65600     iPhone     Luxury          65600             iPhone
## 13     NA    Android Non-Luxury          42100            Android
## 14  94600    Android     Luxury          94600            Android
## 15  59400     iPhone     Luxury          59400             iPhone
## 16  47300     iPhone Non-Luxury          47300             iPhone
## 17  72100       <NA>     Luxury          72100             iPhone
## 18     NA     iPhone Non-Luxury          78200             iPhone
## 19     NA    Android     Luxury          42100            Android
## 20  83000     iPhone     Luxury          83000             iPhone
## 21  64100    Android Non-Luxury          64100            Android
## 22  42100     iPhone Non-Luxury          42100             iPhone
## 23     NA     iPhone     Luxury          59500             iPhone
## 24  91500     iPhone Non-Luxury          91500             iPhone
## 25  51200    Android     Luxury          51200            Android
## 26  13800     iPhone Non-Luxury          13800             iPhone
## 27  47500     iPhone Non-Luxury          47500             iPhone
```

2. EXPLORACIÓN DE PATRONES DE DATOS E INTRODUCCIÓN A LAS TÉCNICAS DE PRONÓSTICOS.

2.1. Estudio de patrones de datos en la serie de tiempo

- El modelo de pronóstico más elaborado fallará si se aplica a datos pocos confiables.
- Cualquier variable integrada con datos recopilados, registrados u observados durante incrementos de tiempos sucesivos se llama *series de tiempo.*

Uno de los pasos más importantes en la selección de un método para pronosticar adecuado con datos de una serie de tiempo es considerar os diferentes tipos de patrones de datos. Existen 4 tipos generales:

1. Horizontal: Cuando los datos reco-pilados en el transcurso del tiempo fluctúan alrededor de un nivel o una media constantes, hay un patrón *horizontal.* Se dice que este tipo de series es *estacionario* en su media.

2. Tendencias: Cuando los datos crecen y descienden en varios períodos, existe un patrón de *tendencia.* La *tendencia* es el componente de largo plazo que representa el crecimiento o el descenso en a serie de tiempo, durante un período extenso.

3. Estacionales: Cuando las observaciones se ven influidas por factores temporales, existe un patrón estaciona.El *componente estacional* se refiere a un patrón de cambio que se repite año trass año.

4. Cíclicos: Cuando las observaciones indican aumentos y caídas que no tienen un período fijo, existe un patrón cíclico. El *componente cíclico* es la fluctuación con forma de onda alrededor de la tendencia y, por lo común, se ve afectada por las condiciones económicas generales. Un componente cíclico, si existe, típicamente presenta un ciclo durante varios años. Las fluctuaciones cíclicas a menudo están influidas por cambios en las expansiones y contracciones económicas.
El *compo0nente cíclico* es la oscilación alrededor de la *tendencia.*

2.2. Exploración de patrones de datos con análisis de autocorrelación.

Cuando se mide una variable a lo largo del tiempo, las observaciones en fdiferentes períodos a menudo están relacionas o correlacionadas. Esta correlación se mide usando el coeficiente de autocorrelación.
Autocorrelación es la correlación que existe entre una variable retrasada uno o más períodos consigo misma.

2.3. Series temporales con R-Studio

Las series temporales se trata de una serie de modelos donde se observan repeticiones a lo largo del tiempo. Los análisis de estas nos ayudarán en cosas como:

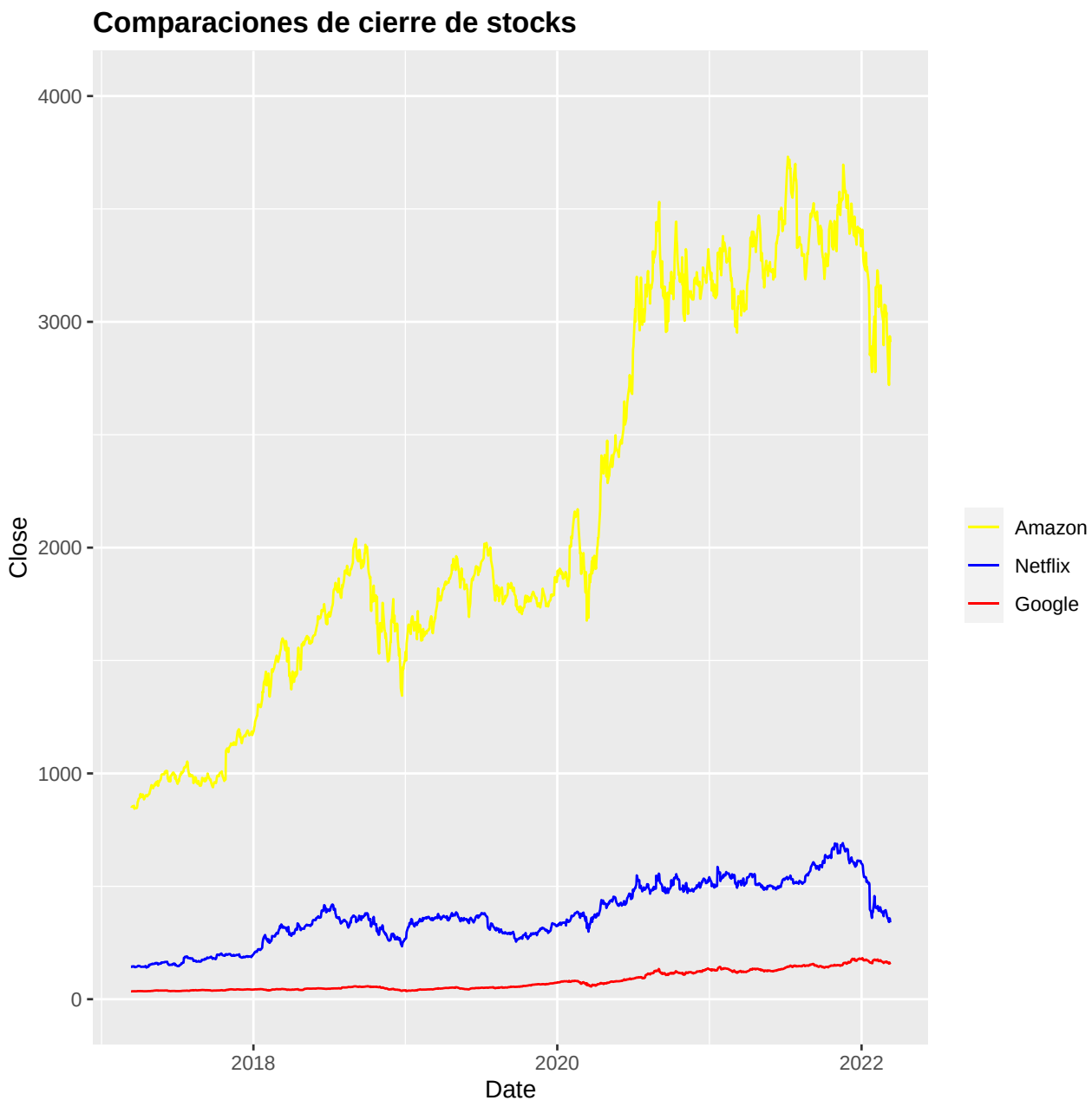
- Analizar y prever el tráfico en una página web.
- Hacer pronósticos de ventas.
- Estudiar el inventario si hay que pedir más de un cierto *ítem* en una determinada época del año o producir menos.

```
AMZN = read.csv("../r-course-master/data/tema6/AMZN_actual.csv", stringsAsFactors = F)
NX = read.csv("../r-course-master/data/tema6/NFLX_actual.csv", stringsAsFactors = F)
GOOG = read.csv("../r-course-master/data/tema6/AAPL_actual.csv", stringsAsFactors = F)
AMZN$Date = as.Date(AMZN$Date)
NX$Date = as.Date(NX$Date)
GOOG$Date = as.Date(GOOG$Date)
```

Gráficos

```
library(ggplot2)
ggplot(AMZN, aes(Date, Close))  +
  geom_line(data=AMZN, aes(color ="Amazon"))+
  geom_line(data=NX, aes(color="Netflix"))+
  geom_line(data=GOOG, aes(color="Google"))+
  labs(color="Legend")+
  scale_color_manual("",
                     breaks = c("Amazon", "Netflix", "Google" ),
                     values = c("yellow", "blue", "red"))+
```

```
  ggtitle("Comparaciones de cierre de stocks")+
  scale_y_continuous(limits = c(0,4000))+
  theme(plot.title = element_text(lineheight = 1, face = "bold"))
```



2.4. El formato fecha en R

```
Sys.Date() #Nos da la fecha de hoy

## [1] "2022-03-28"

as.Date("1/1/80", format="%m/%d/%y") #Años en dígitos

## [1] "1980-01-01"

as.Date("1/1/1980", format="%m/%d/%Y") #Ano en cuatro dígitos

## [1] "1980-01-01"
```

```r
as.Date("2018-01-06") #Formato yyyy-mm-dd o yyyy/mm/dd
```

```
## [1] "2018-01-06"
```

```r
nac <- as.Date("99/11/8")
as.numeric(as.Date("1988/05/19")) #Días que han pasado hasta la fecha
```

```
## [1] 6713
```

```r
#Nombre de los meses
as.Date("Ene 6, 2018", format="%b %d, %Y") #nota en la b e Y
```

```
## [1] NA
```

```r
as.Date("Enero 6, 18", format="%B %d, %y") #Nota en la B e y
```

```
## [1] "2018-01-06"
```

```r
#Fechas desde días de EPOCH
#EPOCH : 1 de Enero de 1970
dt <- 2018
class(dt) <- "Date"
dt
```

```
## [1] "1975-07-12"
```

```r
dt <- -2018
class(dt) <- "Date"
dt
```

```
## [1] "1964-06-23"
```

```r
#Fechas desde días de un punto dado
dt <- as.Date(2018, origin = as.Date("1999-08-11"))
as.Date(-2018, origin = as.Date("1999-08-11"))
```

```
## [1] "1994-01-31"
```

```r
#Componentes de las fechas
dt
```

```
## [1] "2005-02-18"
```

```r
format(dt, "%Y")#Año en 4 dígitos
```

```
## [1] "2005"
```

```r
as.numeric(format(dt, "%Y"))#Año como número en lugar de String
```

```
## [1] 2005
```

```r
format(dt, "%y")#Año en 2 dígitos
```

```
## [1] "05"
```

```r
#Año como número en lugar de String
as.numeric(format(dt, "%y")) #Año como número en lugar de String
```

```
## [1] 5

#Mes como String
format(dt, "%b") #Abreviado

## [1] "feb."

format(dt, "%B") #Nombre completo del mes

## [1] "febrero"

months(dt) #Nos da el mismo resultado de format()

## [1] "febrero"

weekdays(dt) #Nos da el dia

## [1] "viernes"

quarters(dt) #Nos da el 4to trimestre

## [1] "Q1"

julian(dt) #Calendario Juliano

## [1] 12832
## attr(,"origin")
## [1] "1970-01-01"
```

Operaciones de fechas.

```
dt <- as.Date("1/1/2001", format = "%d/%m/%Y")
dt+100 #Sumar 100 días

## [1] "2001-04-11"

dt-100 #Restar 100 días

## [1] "2000-09-23"

dt2 <- as.Date("2001/01/02") #Formto anglosajon
dt2-dt #Diferencia de fechas

## Time difference of 1 days

dt-dt2 #Diferencia de fechas

## Time difference of -1 days

as.numeric(dt2-dt)  #No. excto de la diferencia en dias

## [1] 1

dt<dt2 #Devuelve un Booleano

## [1] TRUE
```

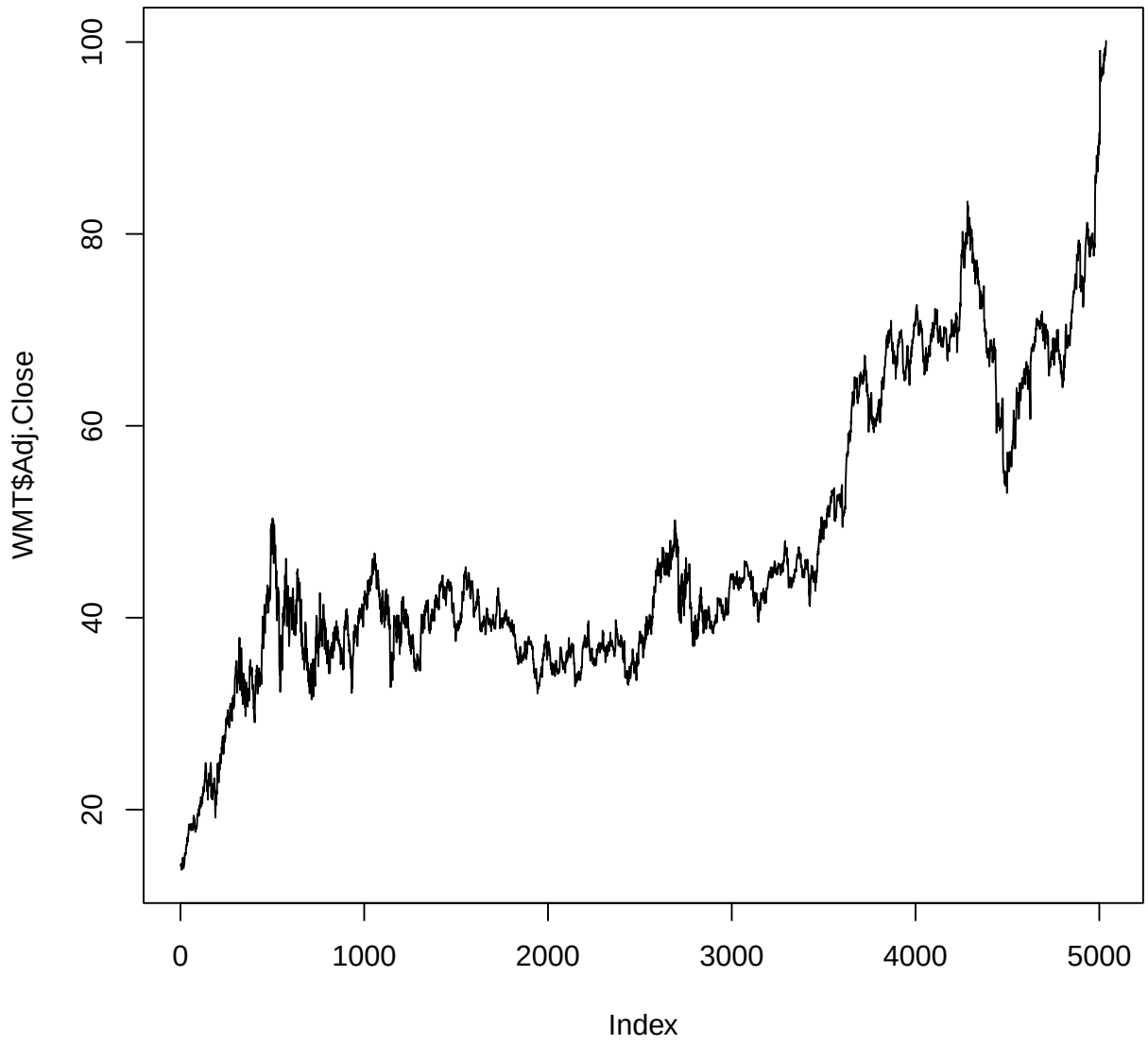
```
dt==dt2 #Devuelve un Booleano

## [1] FALSE

dt2<dt #Devuelve un Booleano

## [1] FALSE
```

Secuencias de fechas.

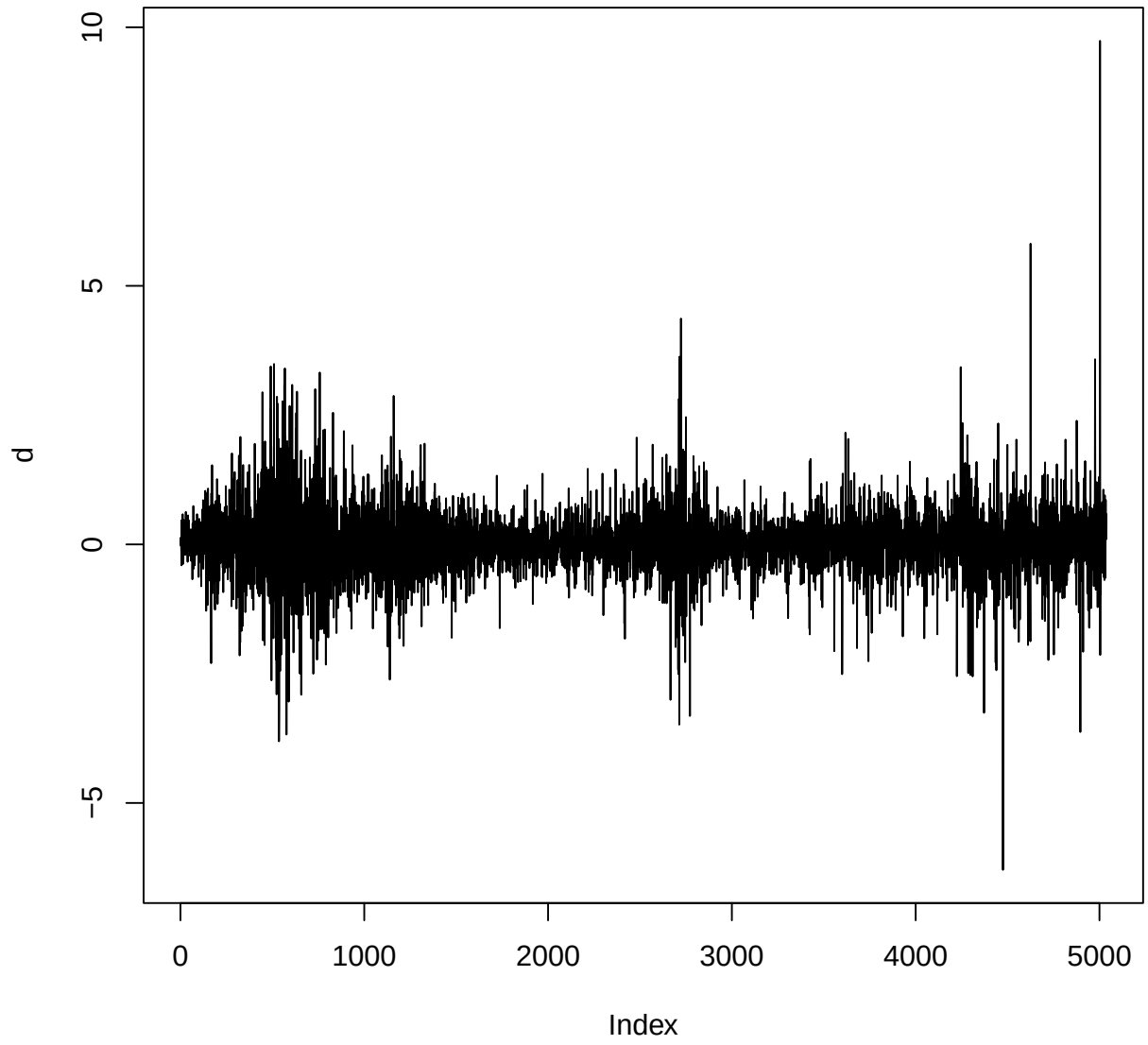
```
seq(dt, dt+180, "month") #Es una secuencia mensual

## [1] "2001-01-01" "2001-02-01" "2001-03-01" "2001-04-01" "2001-05-01"
## [6] "2001-06-01"

seq(dt, as.Date("2001/01/10"), "day") #Es una secuencia diaria.

##  [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
##  [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"

seq(dt, dt+180, "2 months") #Secuencia bimensual.

## [1] "2001-01-01" "2001-03-01" "2001-05-01"

seq(from = dt, by = "4 months", length.out = 6) #Me da los 6 meses con secuencia de 4 meses

## [1] "2001-01-01" "2001-05-01" "2001-09-01" "2002-01-01" "2002-05-01"
## [6] "2002-09-01"

seq(from = dt, by = "4 months", length.out = 6)[3] #Me da los 6 meses con secuencia de 4 meses

## [1] "2001-09-01"
```

2.5. Análisis preliminar de un serie temporal

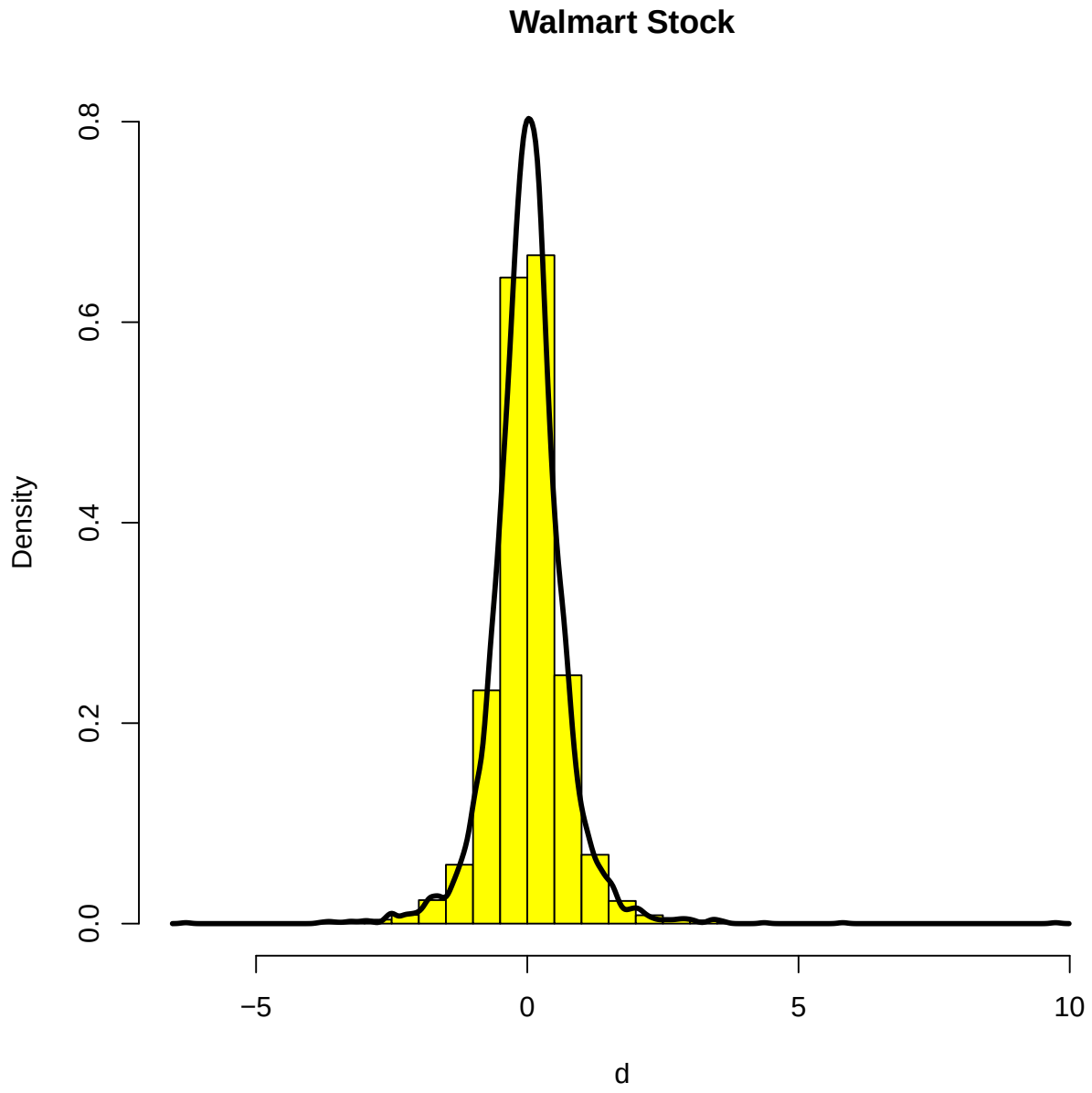
```
WMT = read.csv("../r-course-master/data/tema6/WMT.csv", stringsAsFactors = F)#Cargar datos
plot(WMT$Adj.Close, type = "l") #Gráfico
```




```
d <- diff(WMT$Adj.Close) #Diferencias de un día al siguiente.
plot(d, type = "l") #Gráfico de la diferencias
```




```
hist(d, probability = T, ylim = c(0,0.8), main = "Walmart Stock", breaks = 40, col = "yellow") #
lines(density(d), lwd =3) #Gráfico de densidad
```




```
wmt.m <- read.csv("../r-course-master/data/tema6/WMT-monthly.csv", stringsAsFactors = F)
wmt.m$Date <- as.Date(wmt.m$Date) #Convierte la fecha en tipo Date
wmt.m.ts <- ts(wmt.m$Adj.Close) #Time series, nos da una serie temproal.

wmt.m.ts

## Time Series:
## Start = 1
## End = 243
## Frequency = 1
##   [1]       null  14.304593  16.640038  18.256878  18.194649  19.836435
##   [7]  21.860561  22.746117  21.259741  19.683277  24.916695  27.171595
##  [13]  29.381411  31.059372  31.104519  33.294018  33.260944  30.820621
##  [19]  34.887844  30.585445  32.078522  34.431244  40.808979  41.760128
##  [25]  50.094051  39.706253  35.354893  40.975422  40.210060  41.843861
##  [31]  41.843861  40.163094  34.620220  34.983700  33.021324  37.979061
##  [37]  38.661320  41.387497  36.498245  36.797009  37.755726  37.763027
```

```
##  [43]  35.610352  40.849884  35.113358  36.172966  37.619495  40.364117
##  [49]  42.120674  43.954327  45.441944  44.921642  40.983551  39.692272
##  [55]  40.359928  36.128845  39.287720  36.172909  39.393600  39.651070
##  [61]  37.157234  35.215496  35.407066  38.331852  41.563995  38.826038
##  [67]  39.608311  41.328983  43.738796  41.284634  43.576172  41.194370
##  [73]  39.276798  39.939053  44.173996  44.270420  42.370323  41.426296
##  [79]  39.117424  39.497402  39.244083  39.732948  40.270691  38.881523
##  [85]  39.449146  39.230774  38.639317  37.516296  35.396759  35.464352
##  [91]  36.308582  37.174862  33.867924  33.113590  35.750885  36.695480
##  [97]  35.365494  34.950050  34.381569  35.806572  34.257824  36.859680
## [103]  36.775188  33.973335  34.141304  37.795677  37.765030  35.328083
## [109]  35.389397  36.681561  37.158443  36.112370  37.034889  36.787590
## [115]  37.354305  35.677200  33.875877  34.062424  35.279781  37.378914
## [121]  37.090187  39.773579  38.872139  41.294296  45.665100  45.476070
## [127]  44.449913  46.363930  46.719864  47.559460  44.319466  44.375061
## [133]  44.517986  37.580505  39.271320  41.552307  40.423222  39.893871
## [139]  39.060699  40.221886  41.020176  39.801998  40.280357  44.228928
## [145]  43.337055  43.539299  44.060829  45.307602  43.956799  41.432808
## [151]  39.621159  42.192776  41.327324  44.370701  44.909588  44.843269
## [157]  44.710621  46.741867  43.332310  43.390652  46.154453  46.355934
## [163]  44.904968  44.541607  44.947227  44.174408  48.276928  50.132435
## [169]  50.864414  52.552574  50.599834  52.415543  50.791943  56.749722
## [175]  60.520187  64.608673  63.020149  64.408257  65.473015  62.854797
## [181]  59.547108  61.386971  62.115368  65.669601  68.645859  66.102089
## [187]  66.187584  69.253067  64.845901  66.115715  68.609802  72.417984
## [193]  70.344040  67.147972  67.165947  68.721474  72.133034  69.472519
## [199]  68.354713  66.997963  68.746231  70.088173  69.904869  80.234329
## [205]  78.712860  78.323112  77.355370  75.806969  72.368050  68.863228
## [211]  66.181442  67.161140  60.396515  60.912243  53.772629  55.275711
## [217]  57.586689  62.862343  62.843395  64.880074  63.814411  67.545738
## [223]  70.193588  70.145531  68.674759  69.803024  67.770485  68.167320
## [229]  66.899399  65.058182  69.142593  70.263618  73.824364  77.182686
## [235]  74.812668  79.073265  77.175270  77.730347  86.852264  96.720268
## [241]  98.232292 100.129997 100.129997
```

```
d <- diff(as.numeric(wmt.m.ts)) #Nos da la diferencia de un día
```

```
## Warning in diff(as.numeric(wmt.m.ts)): NAs introducidos por coerción
```

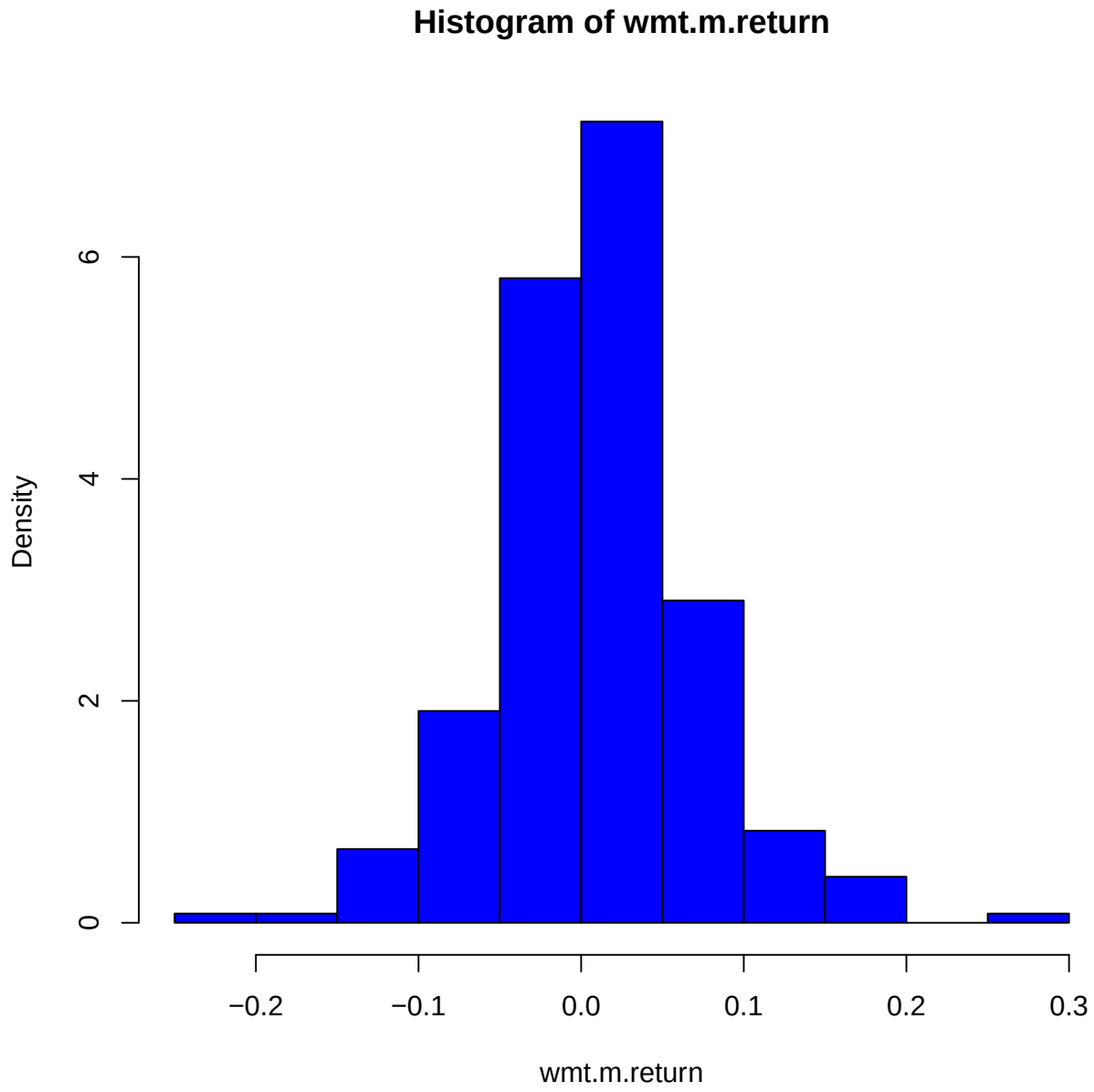
```
d.2 <- diff(as.numeric(wmt.m.ts), lag = 2) #Nos da la diferencia de 2 días
```

```
## Warning in diff(as.numeric(wmt.m.ts), lag = 2): NAs introducidos por coerción
```

```
wmt.m.return <- d / lag(as.numeric(wmt.m.ts), k = -1) #k=-1 nos da un dia hacia atrás, esto nos
```

```
## Warning in lag(as.numeric(wmt.m.ts), k = -1): NAs introducidos por coerción
## Warning in d/lag(as.numeric(wmt.m.ts), k = -1): longitud de objeto mayor no es múltiplo
de la longitud de uno menor
```

```
hist(wmt.m.return, prob = T, col = "blue")
```

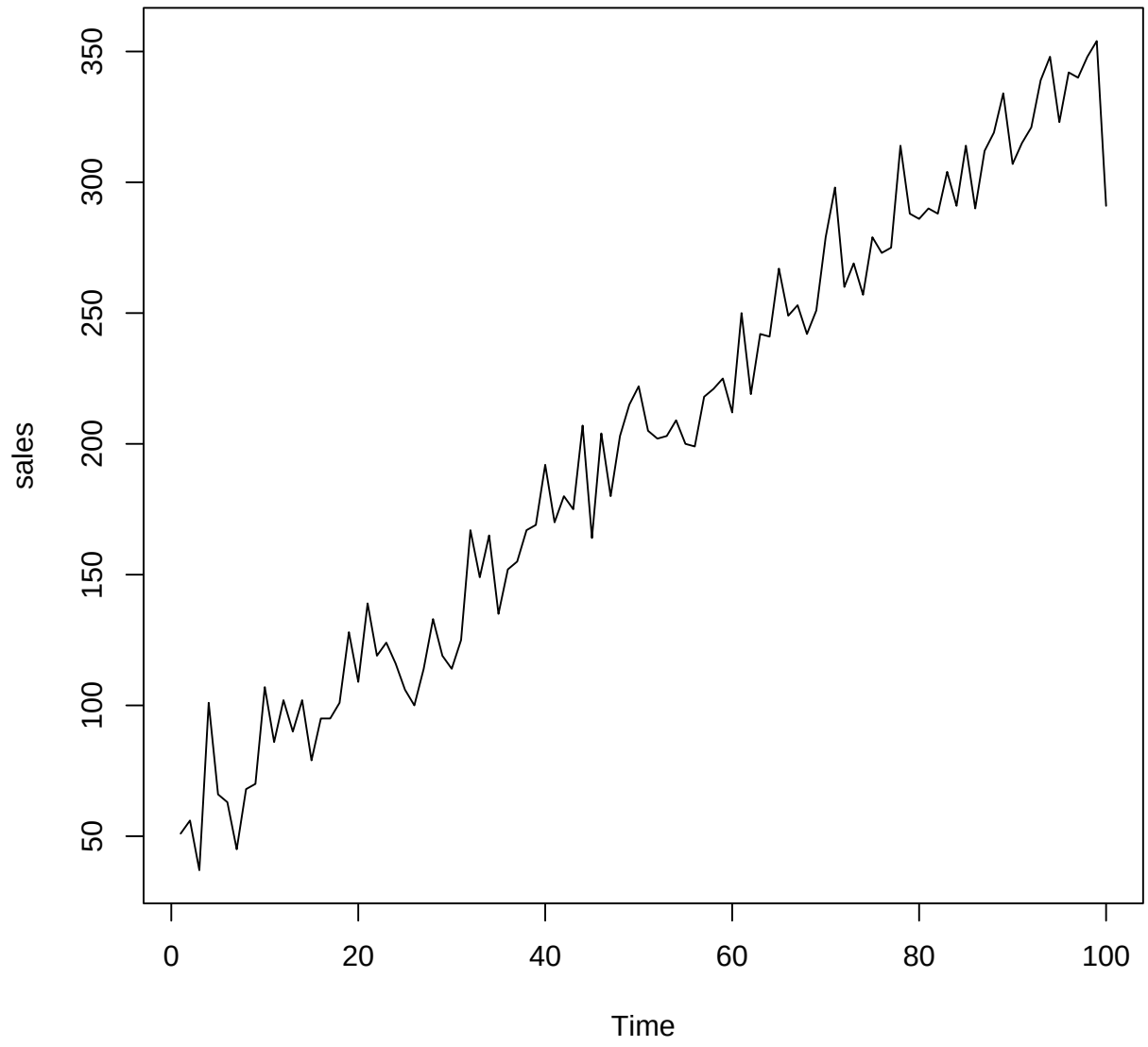


2.6. El objeto serie temporal de R

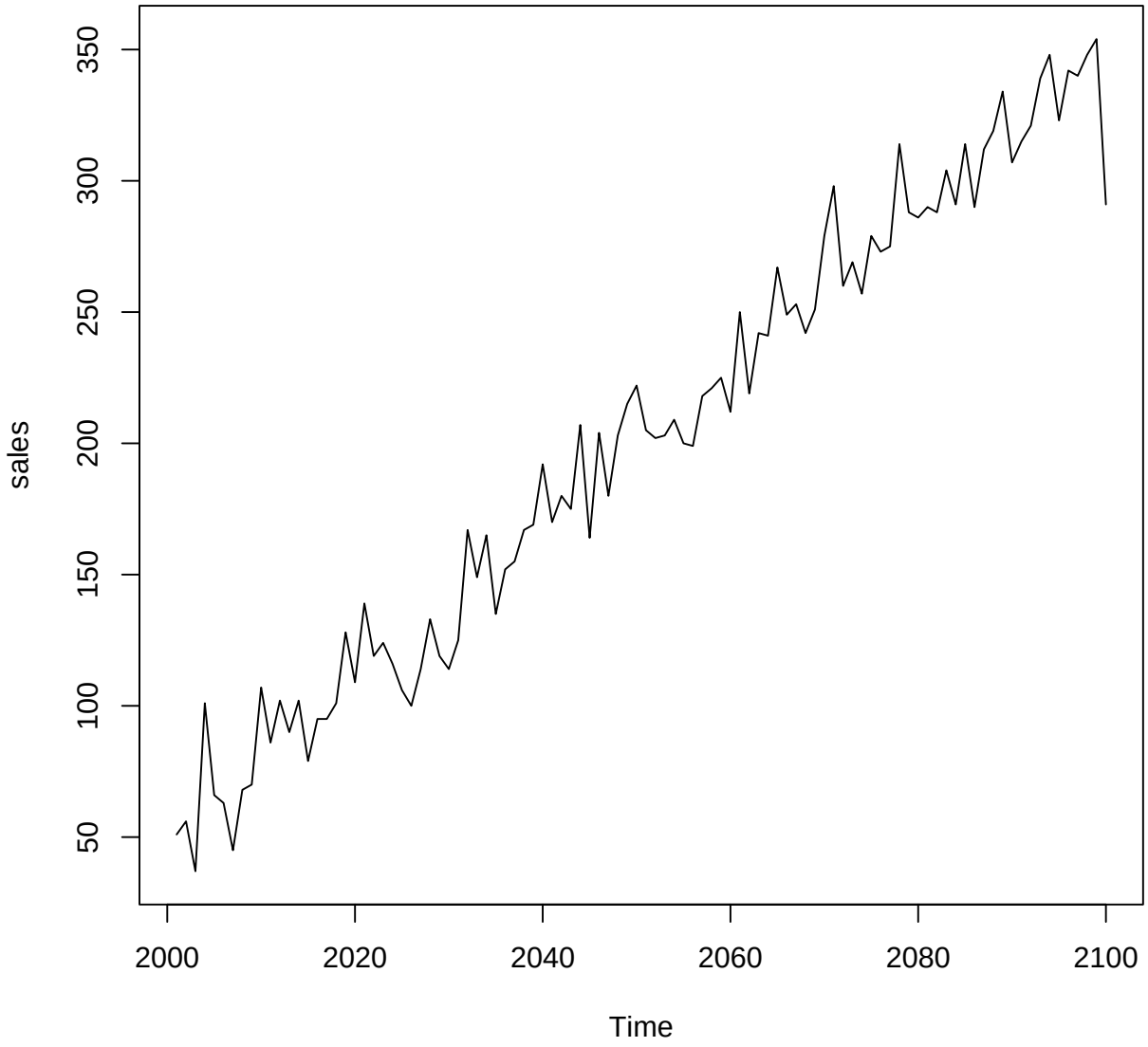
2.6.1. Serie temporal con una columna de datos

```
s <- read.csv("../r-course-master/data/tema6/ts-example.csv")
head(s,1)

##   sales
## 1    51

s.ts <- ts(s) #Serie temporal
class(s.ts)#Nos da que s.ts es de clase Serie temporal

## [1] "ts"

head(s.ts,1)
```

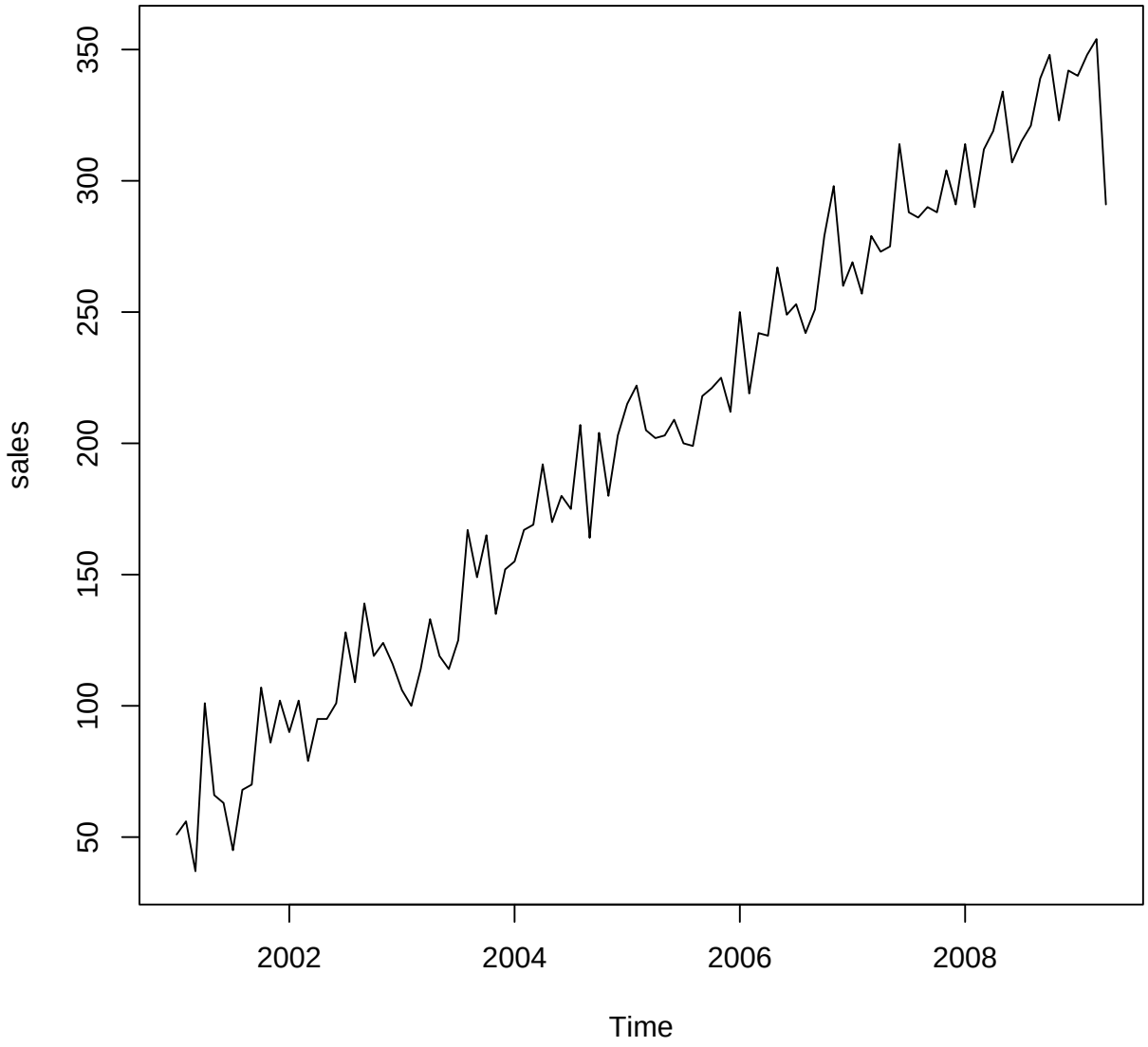
```
##      sales
## [1,]    51

plot(s.ts)
```




```
s.ts.a <- ts(s, start = 2001) #Serie temporal en donde inicia la serie (2001), frecuencia por ai
plot(s.ts.a)
```



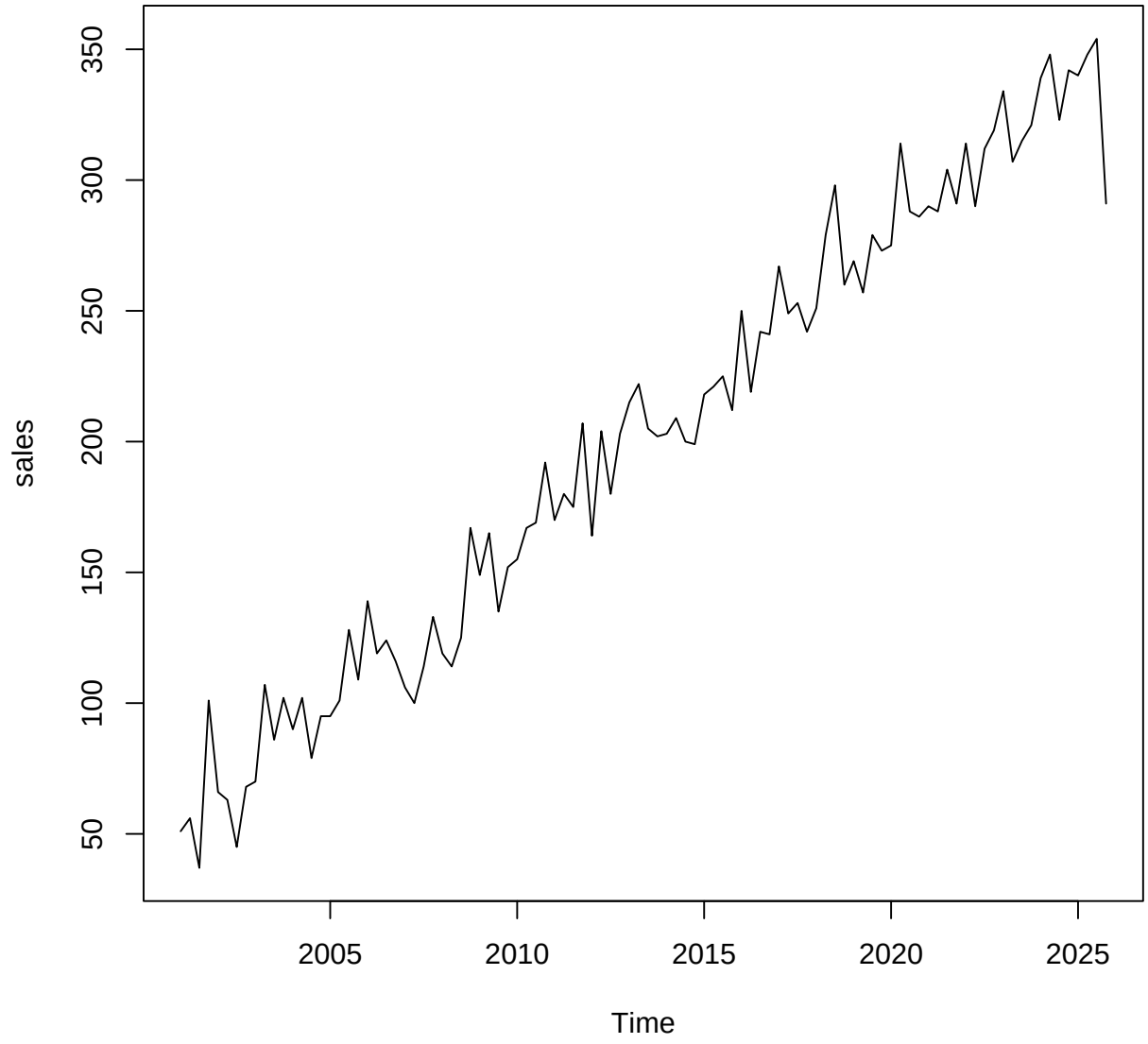

```
 #c(2001,1) indica enero-2001, frecuency=12 indica que tomo 12 muestras anuales
s.ts.m <- ts(s, start = c(2001,1), frequency = 12)
s.ts.m #Nos da la tabla de valores mensuales

##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 2001  51  56  37 101  66  63  45  68  70 107  86 102
## 2002  90 102  79  95  95 101 128 109 139 119 124 116
## 2003 106 100 114 133 119 114 125 167 149 165 135 152
## 2004 155 167 169 192 170 180 175 207 164 204 180 203
## 2005 215 222 205 202 203 209 200 199 218 221 225 212
## 2006 250 219 242 241 267 249 253 242 251 279 298 260
## 2007 269 257 279 273 275 314 288 286 290 288 304 291
## 2008 314 290 312 319 334 307 315 321 339 348 323 342
## 2009 340 348 354 291

plot(s.ts.m)
```




```
s.ts.q <- ts(s, start = 2001, frequency = 4) #4 tomas al año
s.ts.q #Nos da la tabla de valores trimestrales.

##      Qtr1 Qtr2 Qtr3 Qtr4
## 2001   51   56   37  101
## 2002   66   63   45   68
## 2003   70  107   86  102
## 2004   90  102   79   95
## 2005   95  101  128  109
## 2006  139  119  124  116
## 2007  106  100  114  133
## 2008  119  114  125  167
## 2009  149  165  135  152
## 2010  155  167  169  192
## 2011  170  180  175  207
## 2012  164  204  180  203
## 2013  215  222  205  202
## 2014  203  209  200  199
```

```
## 2015  218  221  225  212
## 2016  250  219  242  241
## 2017  267  249  253  242
## 2018  251  279  298  260
## 2019  269  257  279  273
## 2020  275  314  288  286
## 2021  290  288  304  291
## 2022  314  290  312  319
## 2023  334  307  315  321
## 2024  339  348  323  342
## 2025  340  348  354  291
```

```
plot(s.ts.q)
```




```
start(s.ts.q) #Nos indica cuando empieza la serie
```

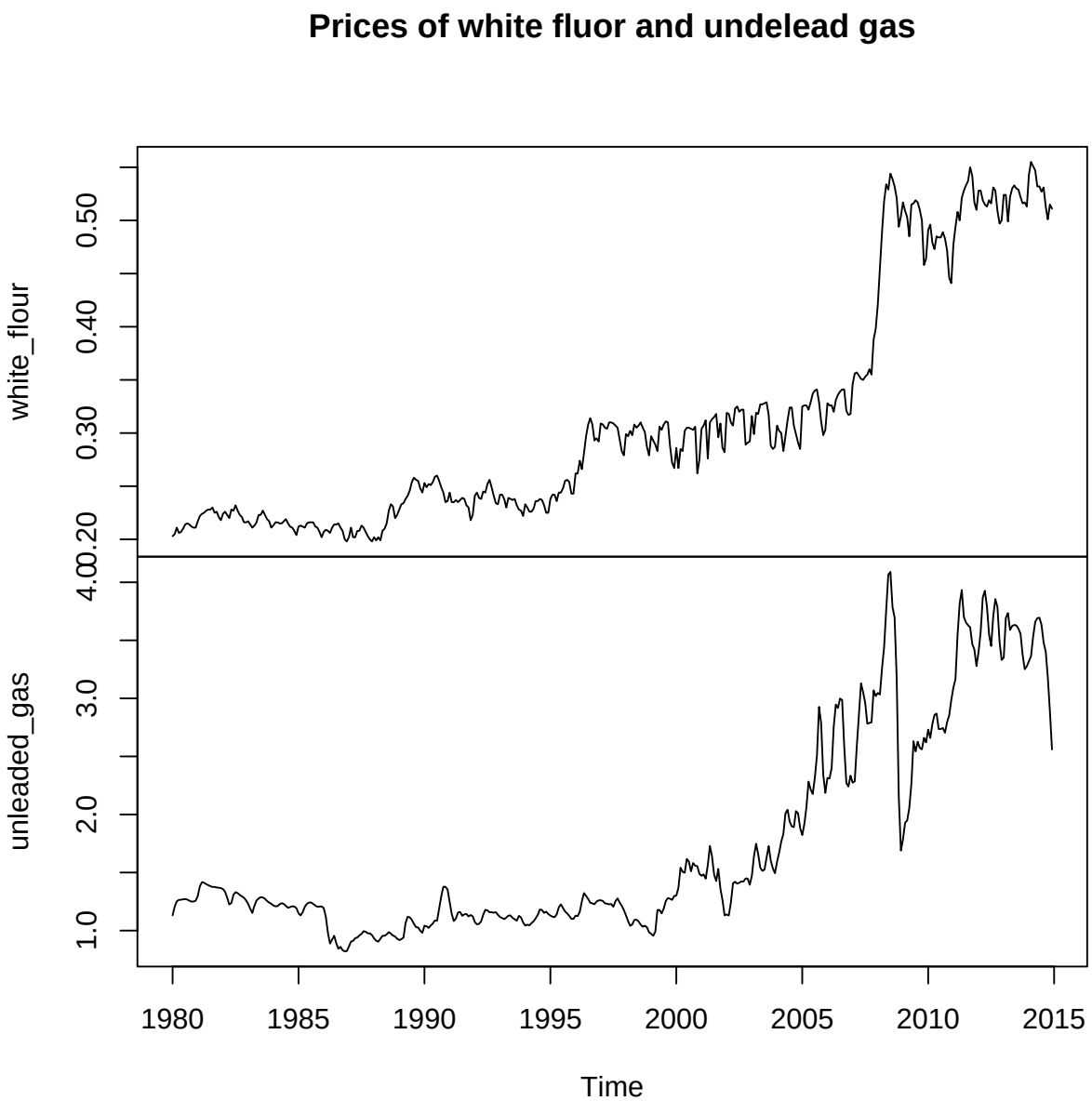
```
## [1] 2001    1
```

```
end(s.ts.q) #Nos indica cuando acaba la serie
```

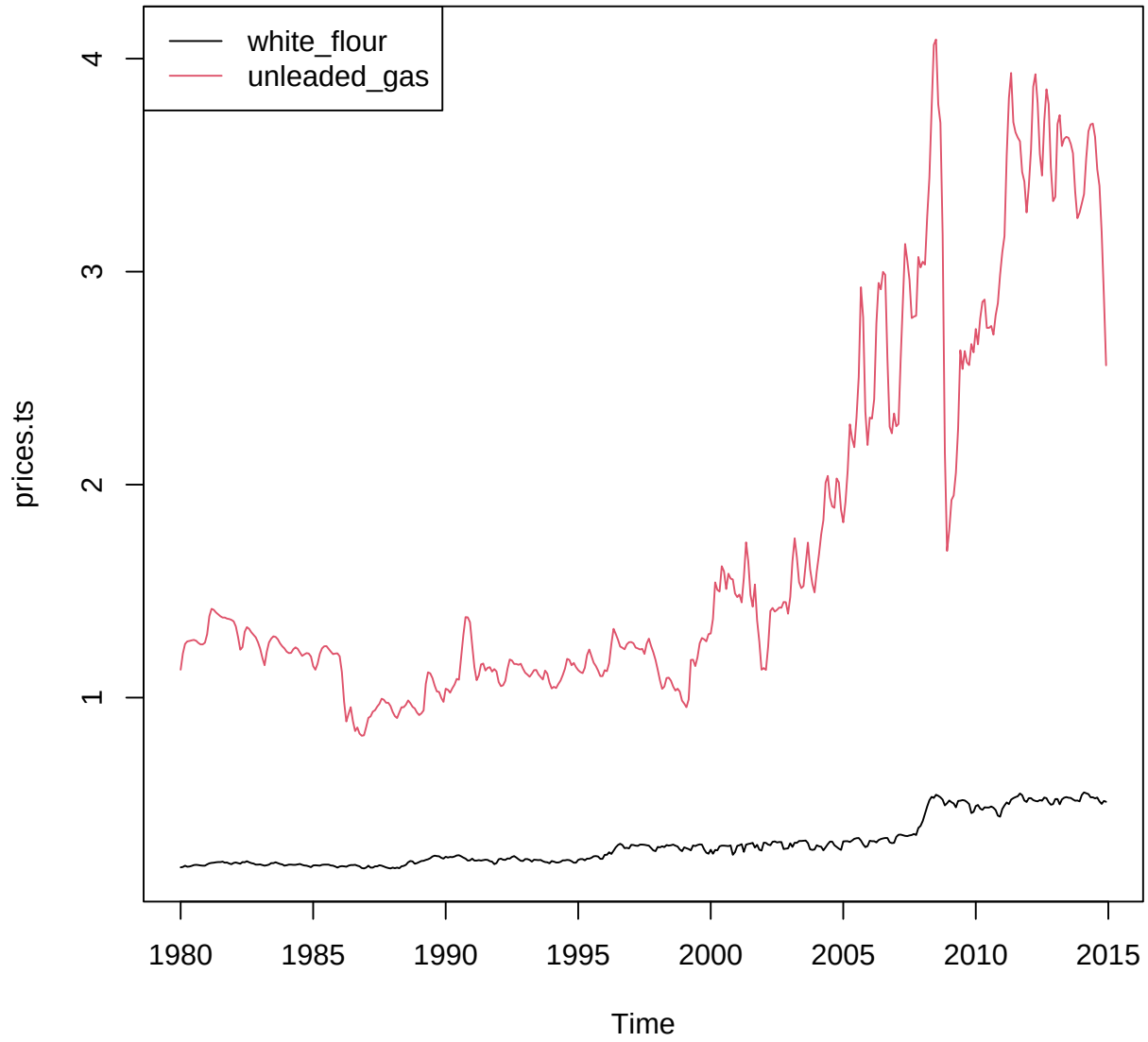
```
## [1] 2025    4

frequency(s.ts.q) #Nos indica el período de la serie

## [1] 4
```

2.6.2. Serie temporal con 2 columnas de datos.

```
prices <- read.csv("../r-course-master/data/tema6/prices.csv")
head(prices,2)

##   white_flour unleaded_gas
## 1       0.203        1.131
## 2       0.205        1.207

prices.ts <- ts(prices, start = c(1980,1), frequency = 12) #Fijamos la fecha de inicio, anual
plot(prices.ts, main = "Prices of white fluor and undelead gas")#Nos da dos gráficos
```



```
plot(prices.ts, plot.type = "single", col = 1:2) #Nos da 1 gráfico con las 2 columnas.
#plot.type = "single", nos junta los gráficos
legend("topleft", colnames(prices.ts), col = 1:2, lty = 1) #Nos dice una leyenda
```



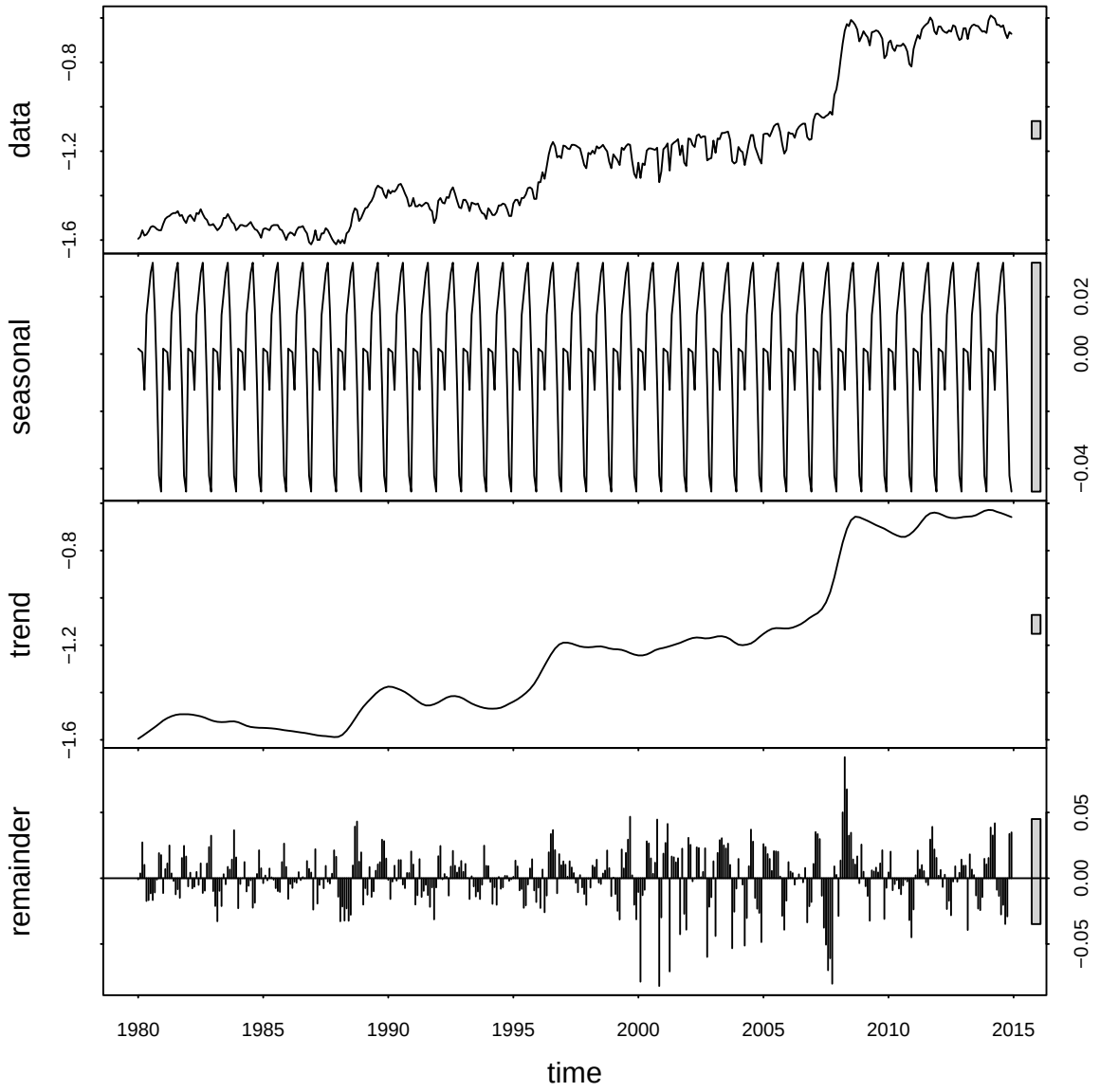

```
#topleft:Arriba a la izquierda, colnames:Con los nombres de la columna. lty=1 : Es el tamaño
```

2.6.3. Descomposición de una serie temporal.

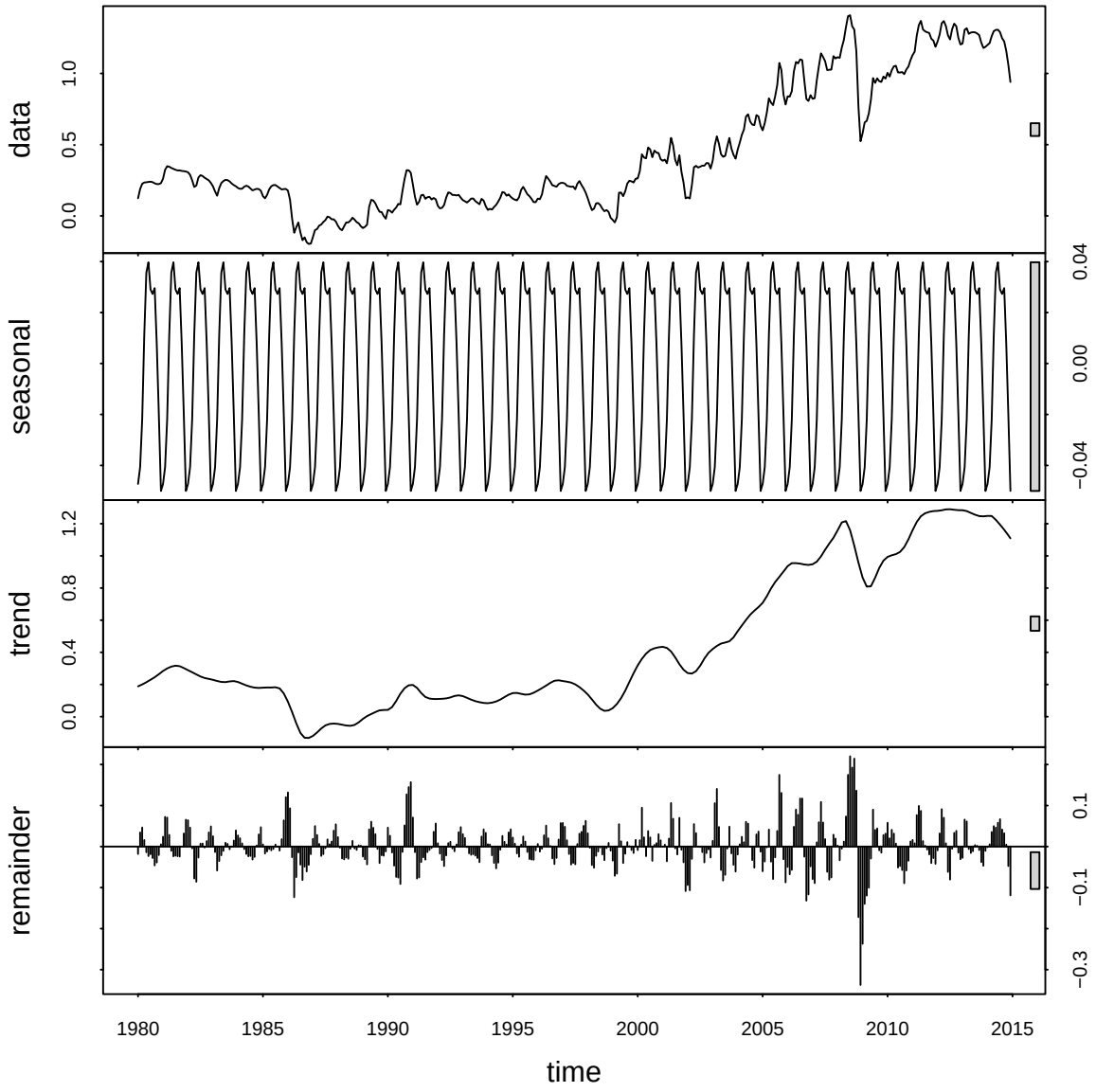
- Una serie temporal se basa en el hecho de que en la mayoría de procesos que existen en el mundo real, tienden a tener lo que llamamos una *estacionalidad* o una *tendencia*
- Es muy útil para poder extraer datos que se van repitiendo a lo largo de diferentes estaciones
- Hay factores que no tienen nada que ver con al *temporalidad*.
- Una **serie multiplicativa** se da cuando la serie va creciendo constantemente, de modo que la amplitud de las fluctuaciones tiene un incremento con el tiempo.
- Aplicar logaritmo cuando una *serie temporal* tenemos la sospecha de que es una *serie multiplicativa* de que cada factor es el anterior multiplicado por un número.

Seasonal Decomposition of Time Series by Loess

```
flour.l <- log(prices.ts[,1]) #Perder el factor multiplicativo
flour.stl<- stl(flour.l, s.window = "period")#Descomposición en estaciones,
#s.window ="period" toma el período original
#data:Valores del dato origina, seasonal:Función períodica ,trend : Tendencia, remainder: Ruido
#data=seasonal+trend+remainder
#flour.stl -> Descompone los valores: seasonal, trend y remainder
plot(flour.stl)
```

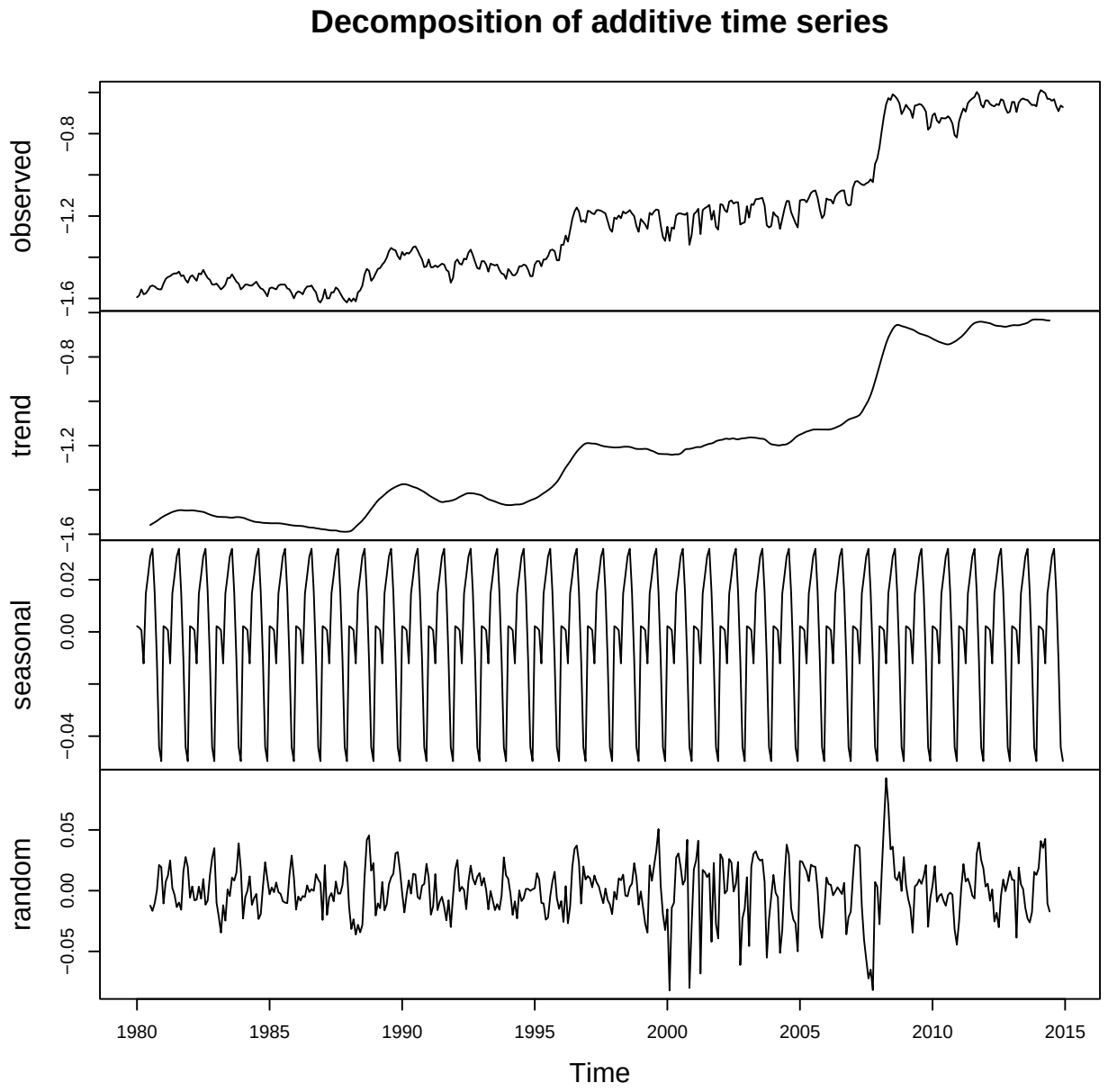



```
gas.l <- log(prices.ts[,2]) #Aplicamos lo mismo para la 2da columna.
gas.stl <- stl(gas.l, s.window = "period")#Aplica el mismo período
plot(gas.stl) #Hace la gráfica
```

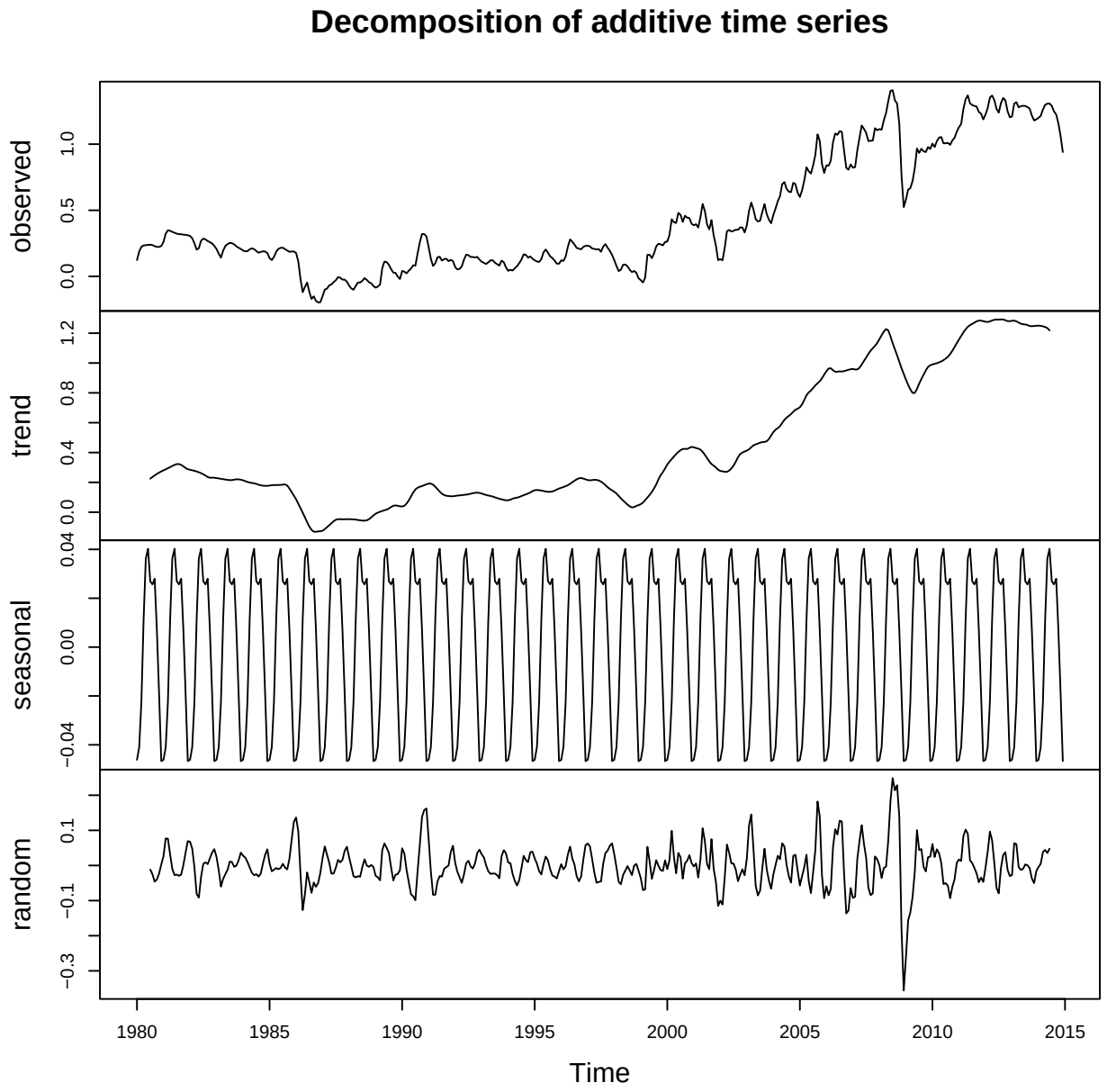


Classical Seasonal Decomposition by Moving Averages

```
flour.dec <- decompose(flour.1) #decompose(): Esta función que nos sirve para descomoponer la se
plot(flour.dec)
```



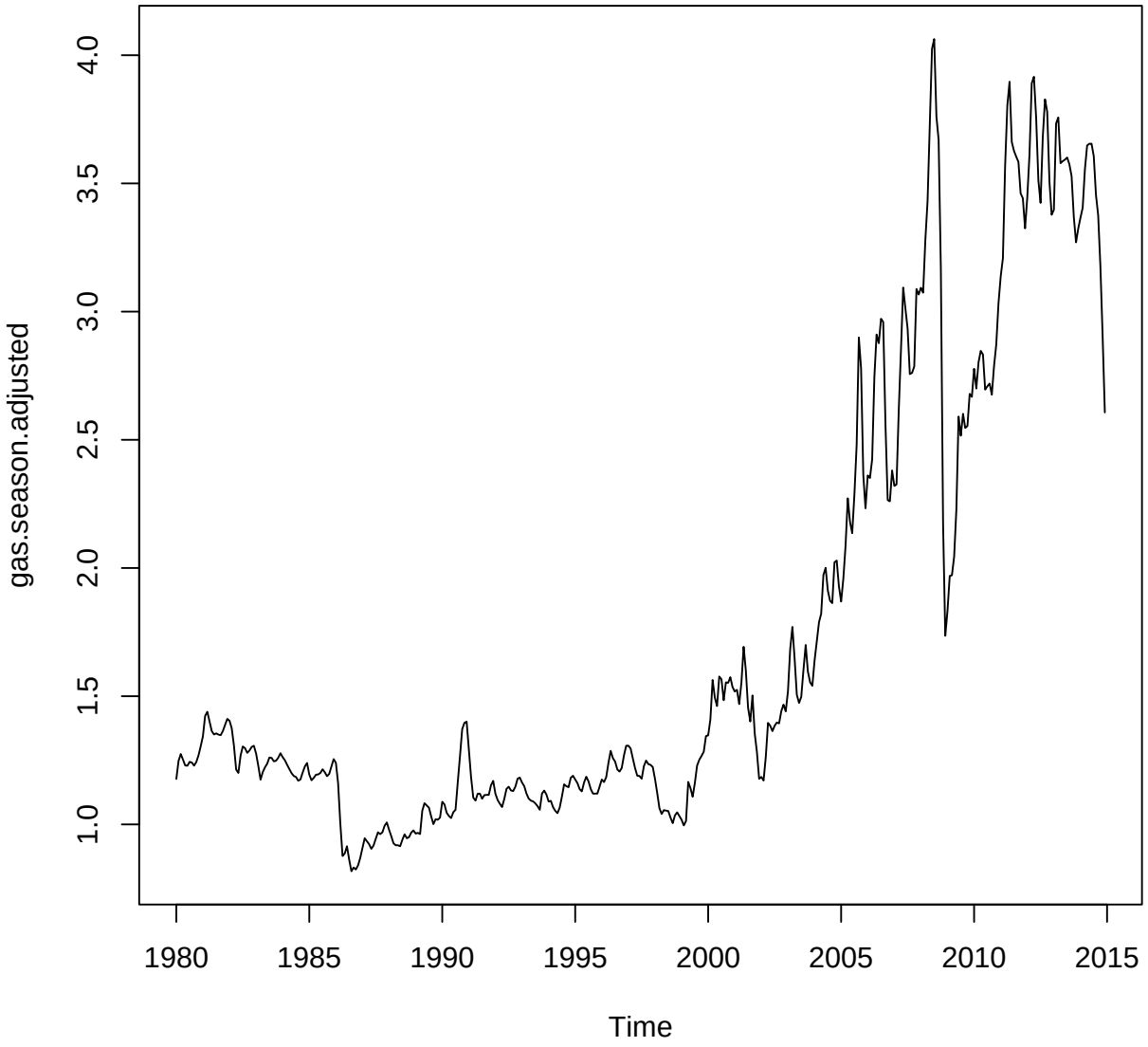

```
gas.dec <- decompose(gas.l)
plot(gas.dec)
```



Ajustar datos originales

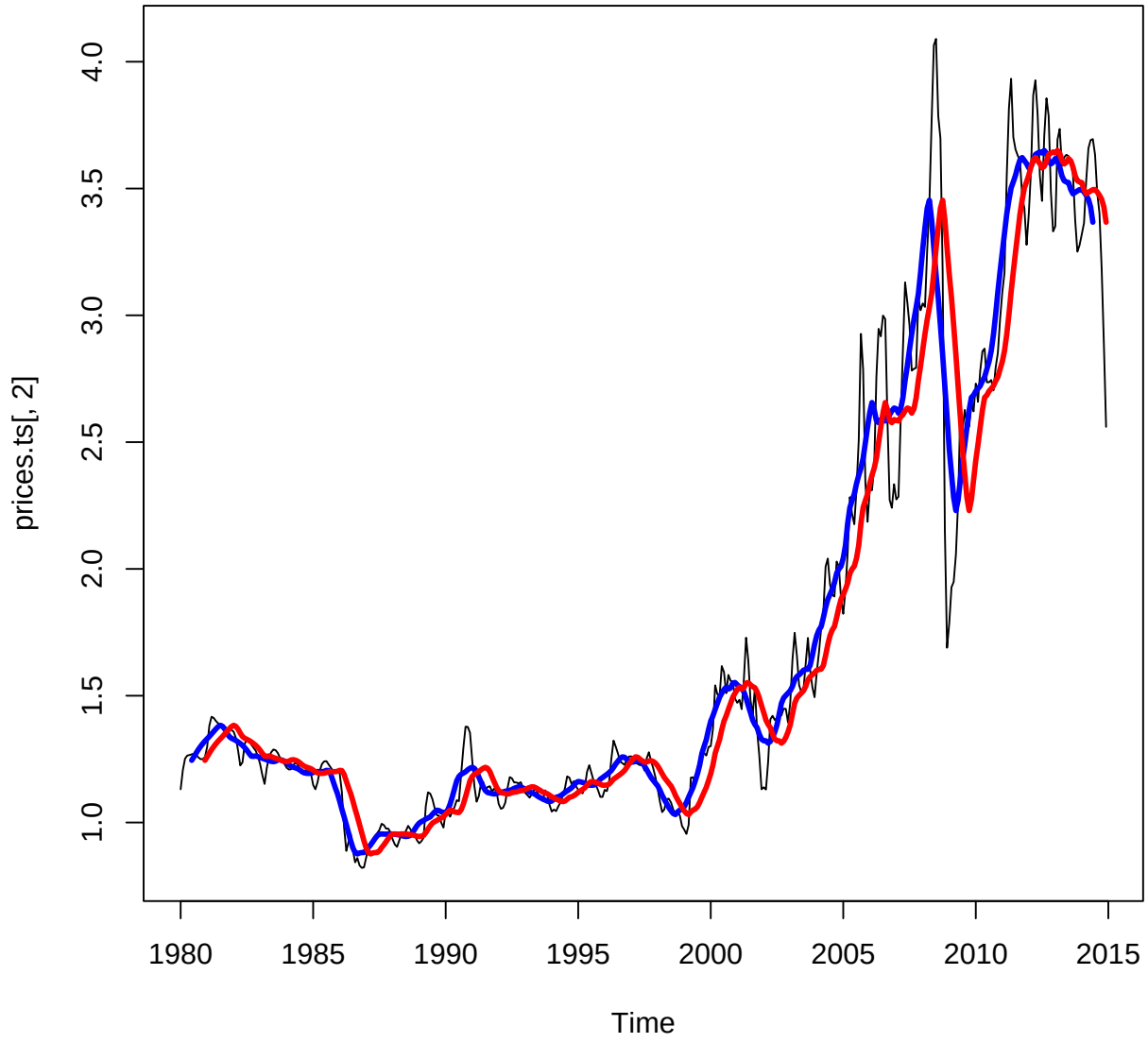
- Podemos tomar los datos originales y ajustarlos en el sentido de eliminar la información que se repite estación tras estación, con la finalidad de obtener cual es la tendencia del mercado.

```
gas.season.adjusted <- prices.ts[,2] - (gas.dec$seasonal)
plot(gas.season.adjusted) #Nos da información con menor ruido
```



De la información original se elimina la infomación que la serie temporal determina que estacional, es decir, que se va repitiendo estación con estación para así tener una visión más clara y completa de cual es la **tendencia global** del mercado.

```
n <- 12 #Perído (En este caso sería anualmente)
gas.f.1 <- filter(prices.ts[,2], filter = rep(1/n, n), sides = 2)
gas.f.2 <- filter(prices.ts[,2], filter = rep(1/n,n), sides = 1)
plot(prices.ts[,2])
lines(gas.f.1, col = "blue", lwd = 3)
lines(gas.f.2, col = "red", lwd = 3)
```



2.7. Suavizado y predicción

2.7.1. Promedios móviles ponderados exponenacialmente(EWMA)

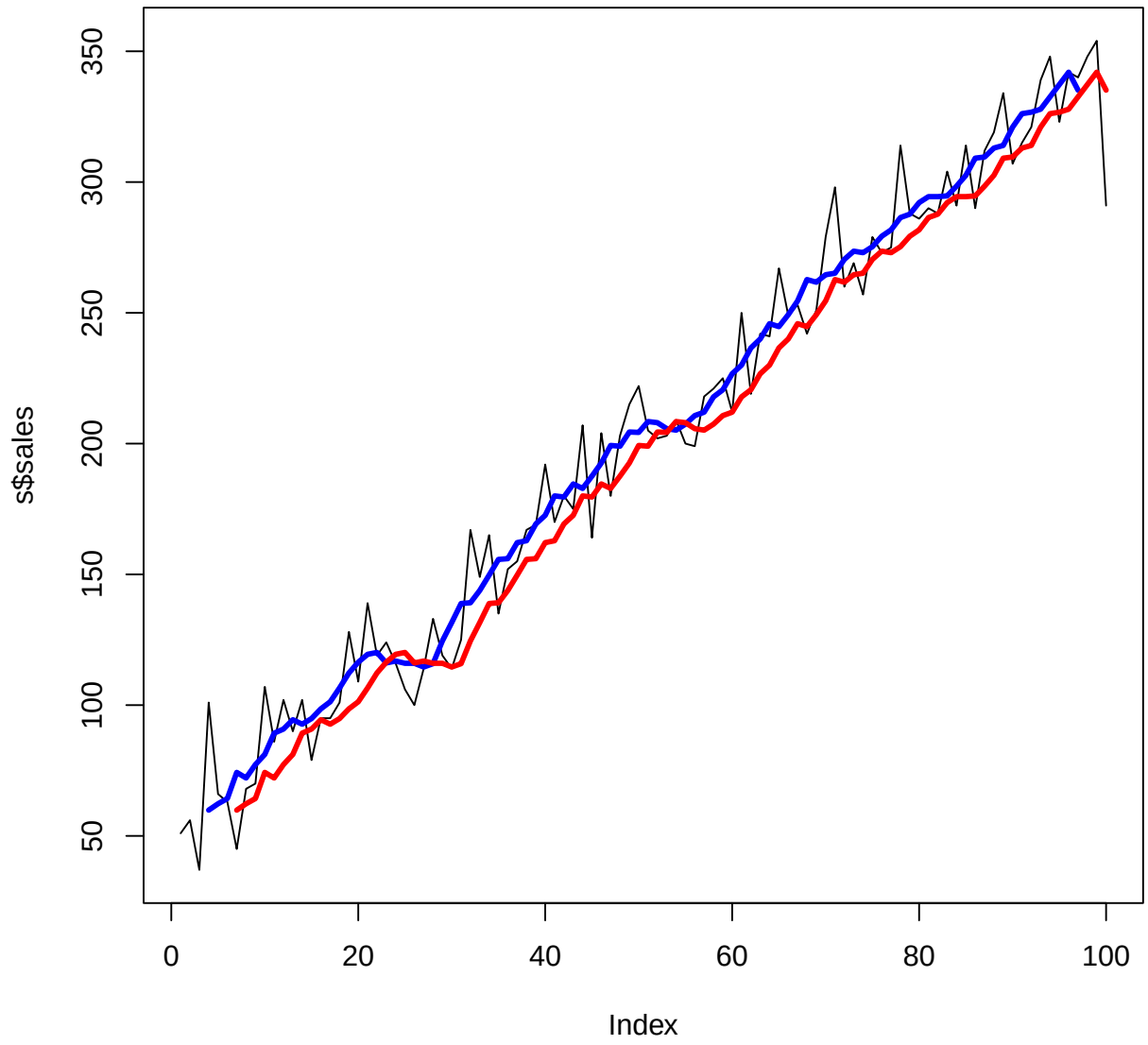
```
s <- read.csv("../r-course-master/data/tema6/ts-example.csv")
s$sales
```

```
##   [1]  51  56  37 101  66  63  45  68  70 107  86 102  90 102  79  95  95 101
##  [19] 128 109 139 119 124 116 106 100 114 133 119 114 125 167 149 165 135 152
##  [37] 155 167 169 192 170 180 175 207 164 204 180 203 215 222 205 202 203 209
##  [55] 200 199 218 221 225 212 250 219 242 241 267 249 253 242 251 279 298 260
##  [73] 269 257 279 273 275 314 288 286 290 288 304 291 314 290 312 319 334 307
##  [91] 315 321 339 348 323 342 340 348 354 291
```

```
plot(s$sales, type = "l")
```

```
n <- 7 #Período (Filtro semanamente)
```

```
weights <- rep(1/n, n) #Pesos
weights

## [1] 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571

#Suavizado de los 3 anteriores con los 3 posteriores del actual
s.fil.1 <- filter(s$sales, filter = weights, sides = 2) #Bi-lateral
lines(s.fil.1, col = "blue", lwd = 3) #Los primeros no se pueden predecir
#Suaviza los 6 anteriores y el actual
s.fil.2 <- filter(s$sales, filter = weights, sides = 1)#Uni-lateral
lines(s.fil.2, col = "red", lwd = 3)
```



Con la técnica de **Moving Avergane**, las tendencias se localizan mucho más fáciles si se promedian con los valores que tenemos cerca de ellos, antes y después con algunas de estas dos técnicas. Lo único que se hace es:

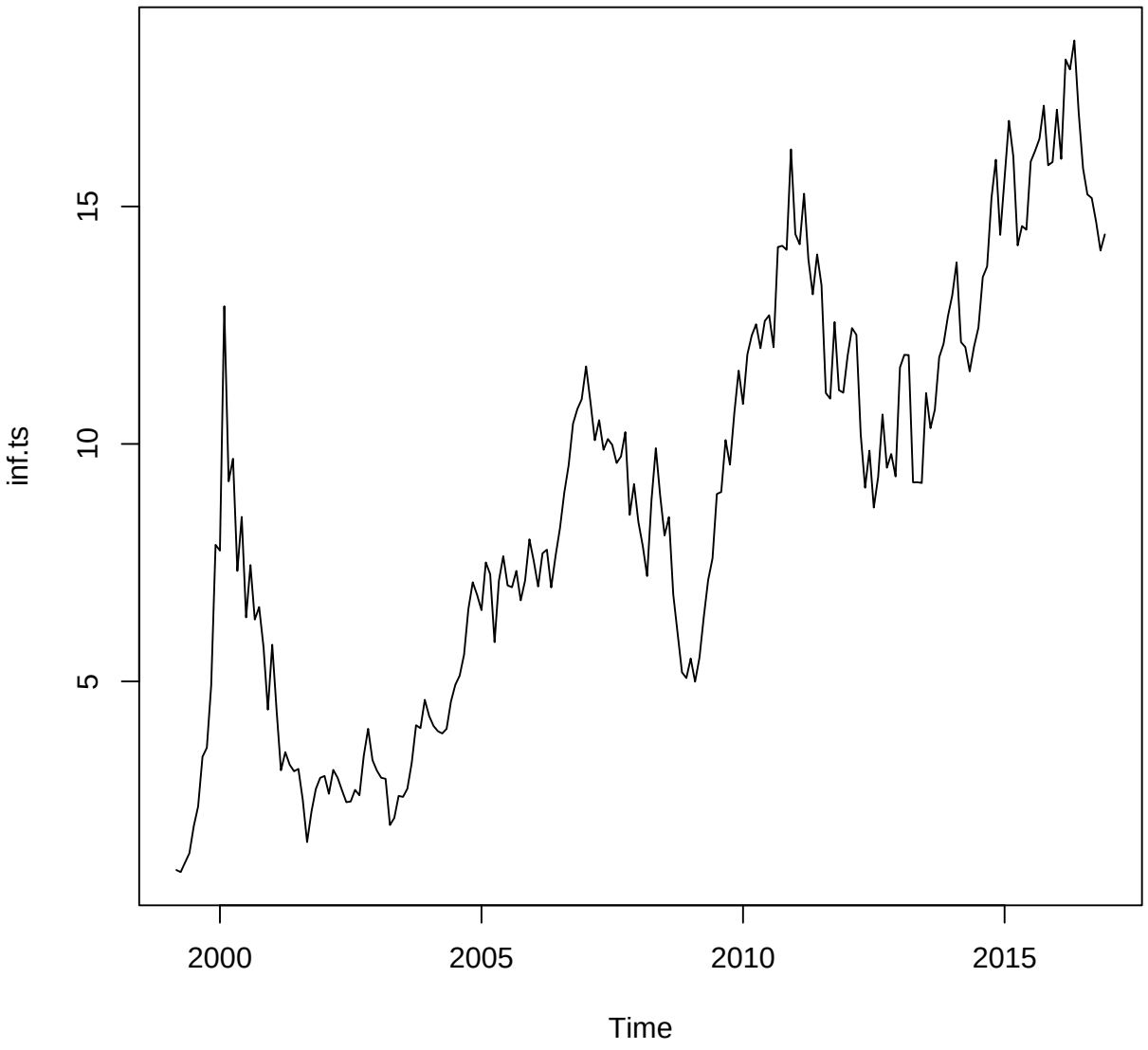
- Crear un filtro de pesos, (tienen que tener la particularidad que sumen 1)

2.7.2. Suavizado exponencial doble (Método de Holt-Winters).

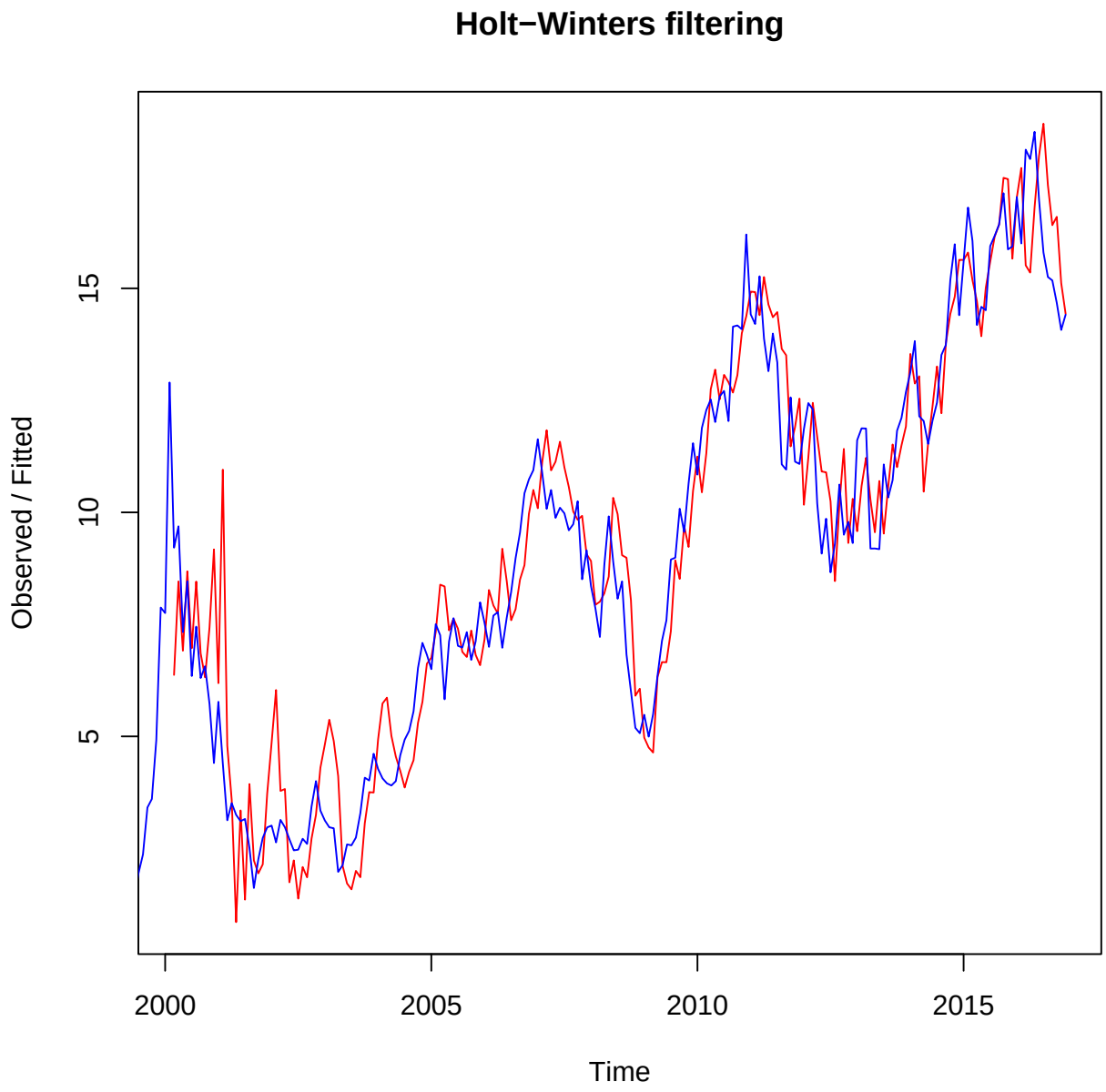
En los anteriores suavizados, no tuvieron en cuenta otros factores que contribuyen, como la *tendencia* y la *estacionalidad*, es decir, este método(Holt-Winters) lleva a cabo un suavizado exponencial en la presencia de tendencias y la estacionalidad

1. Suavizamos las curvas, es decir, eliminamos el ruido de las mismas.
2. Utilizamos el paquete *forecast*, para llevar a cabo una predicción de valores en el futuro.

```
inf <- read.csv("../r-course-master/data/tema6/INFY-monthly.csv")
#head(inf, 3): Quiero ver los primeros 3 datos de la tabla.
#tail(inf): Me da los últimos valores de la tabla
inf.ts <- ts(inf$Adj.Close, start=c(1999,3), frequency = 12)
#inf.ts
plot(inf.ts)
```



```
inf.hw <- HoltWinters(inf.ts)
#head(inf.hw)
plot(inf.hw, col = "blue", col.predicted = "red")
```

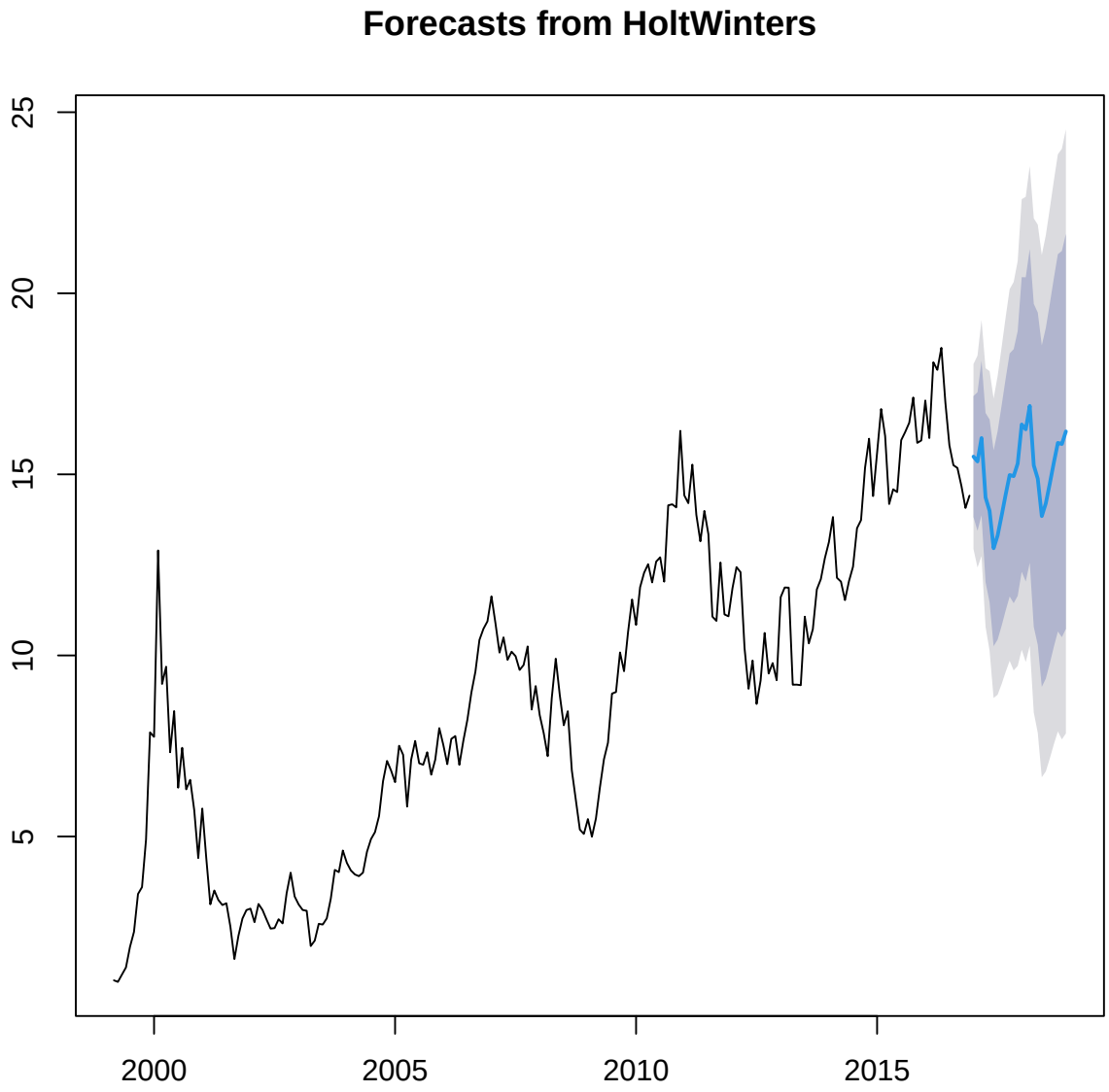



```
library(forecast)

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

inf.hw$SSE #Nos va el valor sse

## [1] 347.1345

inf.hw$alpha #Me da el valor de aalpha

##     alpha
## 0.5519187
```

```
inf.hw$beta #Me da el valor de beta

##       beta
## 0.01042698

inf.hw$gamma #Me da el valor de gamma

## gamma
##     1

head(inf.hw$fitted) #Nos da los primeros valores ajustados

##               xhat    level     trend      season
## Mar 2000 6.371099 5.038675 0.2702327  1.06219146
## Apr 2000 8.459056 6.876835 0.2865814  1.29563975
## May 2000 6.911948 7.842393 0.2936611 -1.22410629
## Jun 2000 8.684473 8.364917 0.2960474  0.02350837
## Jul 2000 6.968464 8.538907 0.2947748 -1.86521779
## Aug 2000 8.455786 8.490957 0.2912012 -0.32637179
```

Predicciones

```
infy.fore <- forecast(inf.hw, h=24)#Predicción a 2 años, (24 por que estabamos en meses)
plot(infy.fore)
```




```
#intervalos de confianza del 85% en tono oscuro y 95% en tono claro
infy.fore$lower #Nos da los valores de predicción

##                 80%       95%
## Jan 2017 13.820929 12.936811
## Feb 2017 13.442656 12.430351
## Mar 2017 13.872663 12.744433
## Apr 2017 12.028455 10.793086
## May 2017 11.468344 10.132484
## Jun 2017 10.255859  8.824740
## Jul 2017 10.437364  8.915217
## Aug 2017 10.819623  9.209944
## Sep 2017 11.239823  9.545552
## Oct 2017 11.626875  9.850518
## Nov 2017 11.442917  9.586633
## Dec 2017 11.645096  9.710761
## Jan 2018 12.310981 10.158075
## Feb 2018 12.042318  9.819248
```

```
## Mar 2018 12.560493 10.268172
## Apr 2018 10.789409  8.428657
## May 2018 10.291257  7.862816
## Jun 2018  9.132117  6.636656
## Jul 2018  9.360123  6.798248
## Aug 2018  9.783316  7.155581
## Sep 2018 10.239846  7.546750
## Oct 2018 10.659357  7.901357
## Nov 2018 10.504566  7.682078
## Dec 2018 10.733080  7.846482
```

```
infy.fore$upper #Zona superior, valores de predicción
```

```
##                80%      95%
## Jan 2017 17.16121 18.04533
## Feb 2017 17.26724 18.27954
## Mar 2017 18.13522 19.26345
## Apr 2017 16.69579 17.93116
## May 2017 16.51534 17.85120
## Jun 2017 15.66275 17.09387
## Jul 2017 16.18817 17.71032
## Aug 2017 16.90113 18.51081
## Sep 2017 17.64093 19.33520
## Oct 2017 18.33811 20.11447
## Nov 2017 18.45613 20.31241
## Dec 2017 18.95319 20.88752
## Jan 2018 20.44485 22.59776
## Feb 2018 20.44127 22.66434
## Mar 2018 21.22109 23.51341
## Apr 2018 19.70854 22.06929
## May 2018 19.46613 21.89457
## Jun 2018 18.56019 21.05566
## Jul 2018 19.03911 21.60099
## Aug 2018 19.71114 22.33887
## Sep 2018 20.41460 23.10770
## Oct 2018 21.07933 23.83733
## Nov 2018 21.16818 23.99067
## Dec 2018 21.63890 24.52550
```

3. Serie de tiempo con pre-procesamiento

```
setwd("C:\\Users\\81799\\Downloads\\tmod_vic-main")
dir()
```

```
##  [1] "descriptor_11_12.csv"    "descriptor_13_20.csv"
##  [3] "FD_2011.xls"             "FD_2012.xls"
##  [5] "FD_2013.xlsx"            "FD_2014.pdf"
##  [7] "FD_2015.pdf"             "FD_2016.pdf"
##  [9] "FD_2017.pdf"             "FD_2018.pdf"
## [11] "FD_2019.pdf"             "FD_2020.pdf"
## [13] "tmod_vic_2011.DBF"       "tmod_vic_2012.DBF"
## [15] "tmod_vic_2013.dbf"       "tmod_vic_2014.dbf"
## [17] "tmod_vic_2015.dbf"       "ts_delitos_por_tipo.pdf"
```

```
library(foreign) #Sirve para leer archivos .dbf
library(tidyverse) #Carpeta para ciencia de datos en R

## - Attaching packages --------------------- tidyverse 1.3.1 -
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v purrr   0.3.4     v forcats 0.5.1
## - Conflicts ---------------------- tidyverse_conflicts() -
## x dplyr::filter()  masks stats::filter()
## x purrr::flatten() masks jsonlite::flatten()
## x dplyr::lag()     masks stats::lag()

tabla1 <- read.dbf("tmod_vic_2011.DBF") #Pero quiero leer todos los archivos juntos
```

3.1. Lectura de todas las tablas (DBF) con for

```
setwd("C:\\Users\\81799\\Downloads\\tmod_vic-main")
indices <- c(13:17) #13,14,15,16 y 17
tablas <- list()
for(i in indices){
  tablas[[i]] <- read.dbf(dir()[i]) #Leer los archivos del 13 al 17
}
dim(tablas[[13]]) #Me dice cuantas filas y columnas tiene el DataFrame

## [1] 27186   138

dim(tablas[[14]]) #Dice cuantas filaas y columnas tiene el DataFrame

## [1] 32493   134
```

3.2. Lapply

```
setwd("C:\\Users\\81799\\Downloads\\tmod_vic-main")
lectura <- function(x){
  read.dbf(dir()[x])
}
tablas2 <- lapply(indices, lectura) #indices =c(13:17)
dim(tablas2[[1]])

## [1] 27186   138
```

```
class(tablas2[[1]]$FAC_DEL) #Nos dice la clase de la columna FAC_DEL

## [1] "factor"

tablas2[[1]]$FAC_DEL[1] #Nos da el 1er valor de la tabla 1, fila 1 columna FAC_DEL

## [1] 446
## 3301 Levels: 100 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 101 ... 999

class(tablas2[[1]]$BPCOD[1])
```

```
## [1] "factor"

#Convierto de clase factor a numeric
tablas2[[1]]$FAC_DEL <- as.numeric(as.character(tablas2[[1]]$FAC_DEL ))
class(tablas2[[1]]$FAC_DEL )

## [1] "numeric"

summary(tablas2[[1]]$BP1_1) #Nos indica los meses que ocurrieron los delitos

##   01   02   03   04   05   06   07   08   09   10   11   12   88   99
## 1569 2128 2171 1680 2058 2623 1936 2132 2143 2365 3110 3148   75   48

tablas2[[1]] <- tablas2[[1]][!(tablas2[[1]]$BP1_1 %in% c("88","99")),] #Me quedo con los meses
summary(tablas2[[1]]$BP1_1)

##   01   02   03   04   05   06   07   08   09   10   11   12   88   99
## 1569 2128 2171 1680 2058 2623 1936 2132 2143 2365 3110 3148    0    0

class(tablas2[[1]]$BP1_1)

## [1] "factor"

tablas2[[1]]$BP1_1 <- as.factor(paste(tablas2[[1]]$BP1_1,2010,sep = "/" )) #aparecera 01/2010,
class(tablas2[[1]]$BP1_1)

## [1] "factor"
```

3.3. Mapply

```
setwd("C:\\Users\\81799\\Downloads\\tmod_vic-main")
 preprocesado <- function(x,y){
   x[,"FAC_DEL"] <- as.numeric(as.character(x$FAC_DEL))#Convierte a numerico
   x <-  x[!(x$BP1_1 %in% c("88","99")),] #Quita las que tengan BP1_1, 88 y 99
   x$BP1_1 <- as.factor(paste(x$BP1_1,y,sep="/")) #Fecha mes/añ
   x #Es como nuestro return (x es la tabla en este caso)
 }
years <- c(2010:2014)
tmod <- mapply(preprocesado,tablas2, years)
length(tmod) #Nos dice cuantos elementos que tiene

## [1] 5

summary(tmod[[1]]$BP1_1) #Aquí nos da el 2010

## 01/2010/2010 02/2010/2010 03/2010/2010 04/2010/2010 05/2010/2010 06/2010/2010
##         1569         2128         2171         1680         2058         2623
## 07/2010/2010 08/2010/2010 09/2010/2010 10/2010/2010 11/2010/2010 12/2010/2010
##         1936         2132         2143         2365         3110         3148

summary(tmod[[2]]$BP1_1) #Aquí nos da el 2011

## 01/2011 02/2011 03/2011 04/2011 05/2011 06/2011 07/2011 08/2011 09/2011 10/2011
##    2025    2682    2636    2028    2317    2637    2116    2402    2368    2707
## 11/2011 12/2011
##    3451    3654
```

```
setwd("C:\\Users\\81799\\Downloads\\tmod_vic-main")
library(tidyverse)
descriptor_1 <- read_tsv("descriptor_11_12.csv", col_names = T)

## Rows: 14 Columns: 2
## - Column specification ----------------------------
## Delimiter: "\t"
## chr (2): CODIGO, DESCRIPCION
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

descriptor_2 <- read_tsv("descriptor_13_20.csv", col_names = T)

## Rows: 15 Columns: 2
## - Column specification ----------------------------
## Delimiter: "\t"
## chr (2): CODIGO, DESCRIPCION
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

dim(tmod[[1]])

## [1] 27063   138

tmod[[1]]$BPCOD[1]

## [1] 02
## Levels: 01 02 03 04 05 06 07 08 09 10 11 12 13 14

tmod[[1]] <- left_join(tmod[[1]],descriptor_1, by = c("BPCOD" ="CODIGO"))
#Al aplicar left_join() es importante que sean de las mismas clases
class(tmod[[1]]$BPCOD)

## [1] "character"

class(descriptor_1$CODIGO)

## [1] "character"

dim(tmod[[1]]) #Se agrega una columna con el nombre de descipción

## [1] 27063   139
```

```
tmod[[2]] <- left_join(tmod[[2]],descriptor_1, by = c("BPCOD" ="CODIGO"))
tmod[[3]] <- left_join(tmod[[3]],descriptor_2, by = c("BPCOD" ="CODIGO"))
tmod[[4]] <- left_join(tmod[[4]],descriptor_2, by = c("BPCOD" ="CODIGO"))
tmod[[5]] <- left_join(tmod[[5]],descriptor_2, by = c("BPCOD" ="CODIGO"))
```

3.4. Tapply

Quiero fijarme por tipo de delito y por mes cuantos delitos se cometieron

```
#tapply sirve para hacer cálculos discriminando
del_2010 <-tapply(tmod[[1]]$FAC_DEL,list(tmod[[1]]$BP1_1 ,tmod[[1]]$DESCRIPCION), function(x){
   sum(x, na.rm = T)
})
del_2010 <- data.frame(rownames(del_2010), del_2010, TOTAL = rowSums(del_2010, na.rm = TRUE))
del_2010
```

```
##              rownames.del_2010. Agr_Fis   Amen    Ext   Fr_B   Fr_C  Hos_S
## 01/2010/2010       01/2010/2010   70230  85385 245756  56955  71850  50137
## 02/2010/2010       02/2010/2010   96353 104262 410498  94557  77040  46877
## 03/2010/2010       03/2010/2010   78964 109599 424317  88604  79622  53580
## 04/2010/2010       04/2010/2010   64842 121895 339422  52496  70104  40405
## 05/2010/2010       05/2010/2010   65898 124840 437295  70488  54332  96767
## 06/2010/2010       06/2010/2010  115559 165510 618637  87871  97116  47516
## 07/2010/2010       07/2010/2010   47570 107707 420127  73659  56875  22688
## 08/2010/2010       08/2010/2010   98186 129341 480523  86887  87382  59704
## 09/2010/2010       09/2010/2010  144103 175413 445688  92440  87414  34333
## 10/2010/2010       10/2010/2010  108805 179364 478331  94806  93562  67033
## 11/2010/2010       11/2010/2010  134299 267006 643469 105570 116568  94940
## 12/2010/2010       12/2010/2010  154719 249842 524302  96292 105874 110147
##                  O_R Otros R_call  R_cas R_p_veh R_t_veh   Sec Vio_S   TOTAL
## 01/2010/2010  92804 17561 278813 104359  134364   37583  2723  3211 1251731
## 02/2010/2010 100378 19130 425417 136996  188933   34825  7879   558 1743703
## 03/2010/2010 103440 21702 419158 129562  195342   42767  5140   448 1752245
## 04/2010/2010  92143 25916 319260 101353  138640   38798   611   164 1406049
## 05/2010/2010  85808 20819 420240 116049  174195   30865  6186  2793 1706575
## 06/2010/2010 151853 26442 462150 174909  223569   31488  7651   840 2211111
## 07/2010/2010  88539 30497 396331 119606  180594   31790 15436   755 1592174
## 08/2010/2010  87497 30584 416537 103546  194254   35046 19424    NA 1828911
## 09/2010/2010 106006 20822 471568 113817  198895   26679  4502  2267 1923947
## 10/2010/2010 129461 23604 435171 152976  219623   31475 22042  1624 2037877
## 11/2010/2010 165697 37630 691967 181641  287167   55120 12937  2339 2796350
## 12/2010/2010 217324 45235 872855 163438  224132   38191  9969   126 2812446
```

3.5. Serie de tiempo

```
library(base)
library(zoo)
```

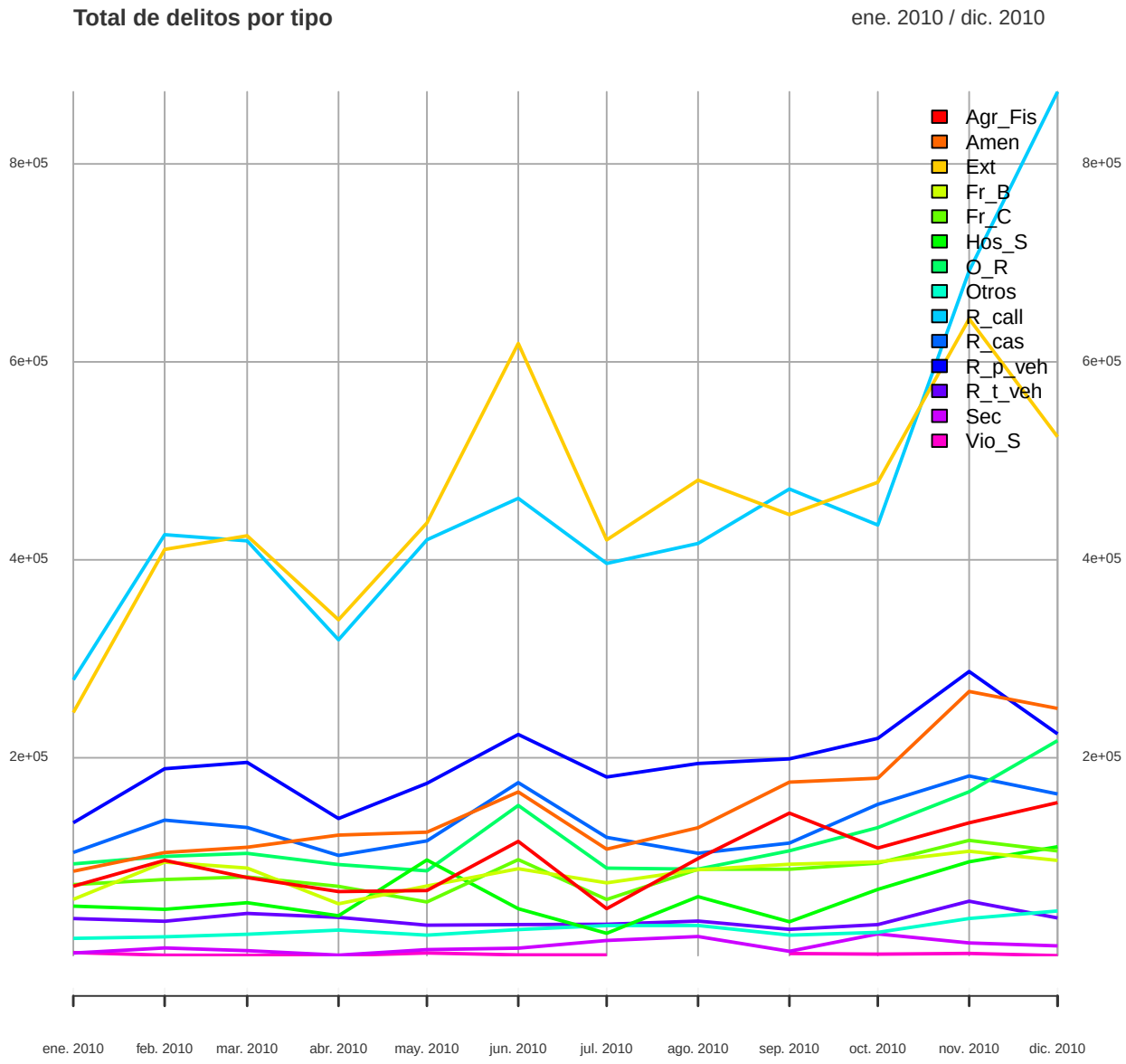
```
##
## Attaching package: 'zoo
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(tidyverse)
library(dplyr)
library(xts)
```

```
##
## Attaching package: 'xts
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
# del_2010[,-1] Quitar la 1er columna
# as.yearmon() es una función que solo acepta meses y año
del_2010 <- xts(del_2010[,-1],
                order.by =as.yearmon(del_2010[,1], format = "%m/%Y") )
```

```
color <- rainbow(ncol(del_2010))
plot.xts(del_2010[,-15], col = color,
         legend.loc = "topright",
         cex = 0.5,
         cex.axis =0.7,
         main = "Total de delitos por tipo")
```



3.6. Creación de pdf

```
pdf("ts_delitos_por_tipo.pdf")
plot.xts(del_2010[,-15], col = color,
         legend.loc = "topright",
         cex = 0.5,
         cex.axis =0.7,
         main = "Total de delitos por tipo")
dev.off()

## pdf
##   2
```