PRÁCTICA 2: ANÁLISIS EXPLORATORIO Y DESCOMPOSICIÓN DE SERIES TEMPORALES

PROCESOS ESTOCÁSTICOS Y SERIES TEMPORALES

GRADO EN CIENCIA E INGENIERÍA DE DATOS

Sumario: En esta práctica veremos cómo definir y representar gráficamente series temporales, así como dos métodos usuales de descomposición de series: la descomposición clásica y la descomposición STL. Se trata de herramientas básicas para el análisis descriptivo de series temporales que sirven como punto de partida dentro del campo de la predicción de series (Forecasting).

1. Definición y representación gráfica de series temporales

Las series temporales se pueden definir usando diferentes tipos de objetos: pueden ser vectores numéricos (usualmente variables de un dataframe), o bien **objetos de clase "ts"** (time series). Este último tipo de objetos contienen atributos adicionales tales como frecuencia y fechas. Es importante tener en cuenta este aspecto, pues muchas funciones de R para el análisis de series temporales requieren que el objeto al que se aplican sea de clase "ts".

Vamos a considerar los datos de una serie temporal incluida en R. Se trata del conjunto de datos "AirPassengers", que contiene el número de pasajeros mensuales en aerolíneas internacionales, desde enero de 1949 hasta diciembre de 1960.

```
# help("AirPassengers")
```

Comenzaremos importando los datos de pasajeros contenidos en el fichero "Pasajeros.csv". La serie de interés se corresponde con la variable "pasajeros" del dataframe importado, se trata por tanto de un vector (en este caso vector numérico de tipo entero)

```
datos_df <- read.csv2(file = "Pasajeros.csv")
class(datos_df)
```

```
## [1] "data.frame"
```

```
class(datos_df$pasajeros)
```

```
## [1] "integer"
```

La función *ts()* de R convierte un objeto a serie temporal. Hay que especificar la frecuencia, en este caso "frequency = 12", lo que indica que a cada unidad temporal le corresponden 12 observaciones. Es decir, en este caso R asume que la unidad temporal a considerar es el año y asume estacionalidad de periodo "L = 12". Además especificaremos que la primera observación corresponde a enero de 1949 con el argumento "start".

```
datos_ts <- ts(datos_df$pasajeros, frequency = 12, start = c(1949, 1))
datos_ts
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
```

```
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

```
class(datos_ts)
```

```
## [1] "ts"
```

IMPORTANTE: Los objetos de la clase "ts" requieren que los datos de la serie temporal estén equiespaciados en el tiempo; por ejemplo, datos horarios, semanales o mensuales. No admite el caso de datos irregularmente distribuidos en el tiempo.

Si recuperamos los datos directamente de R, podemos comprobar que ya se trata de un objeto de clase "ts". Podemos pedir que nos muestre los datos de la serie, su frecuencia (periodo), así como fechas de inicio y fin.

```
datos <- AirPassengers
class(datos)
```

```
## [1] "ts"
```

```
datos
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

```
frequency(datos)
```

```
## [1] 12
```

```
start(datos)
```

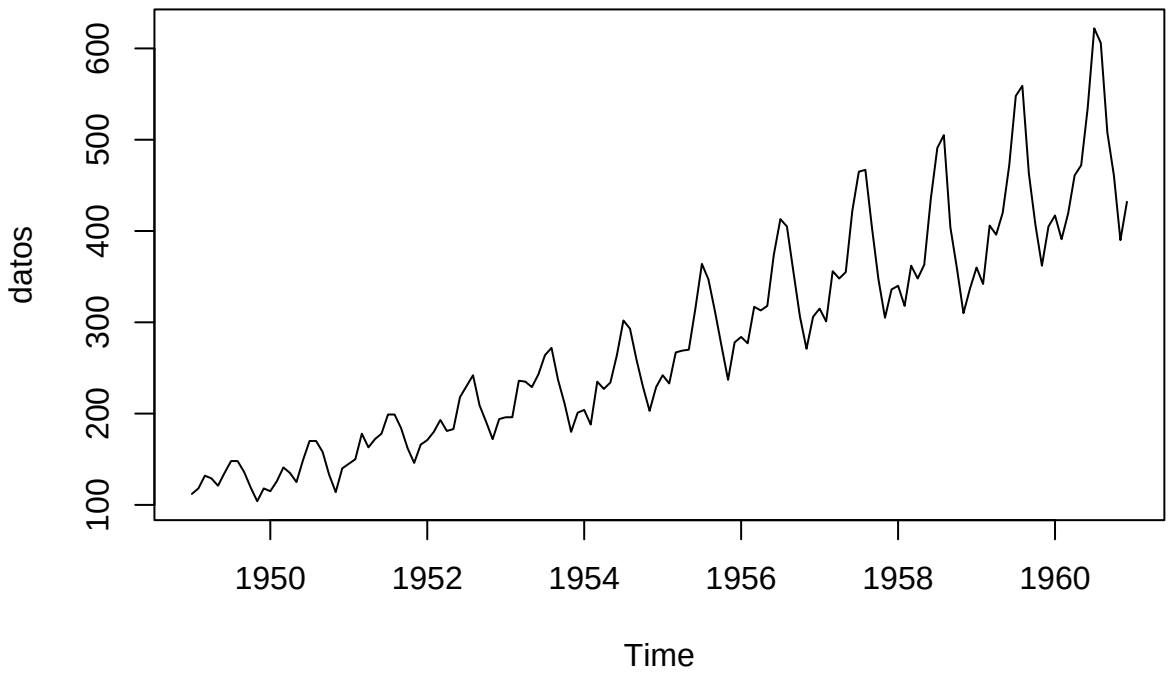
```
## [1] 1949    1
```

```
end(datos)
```

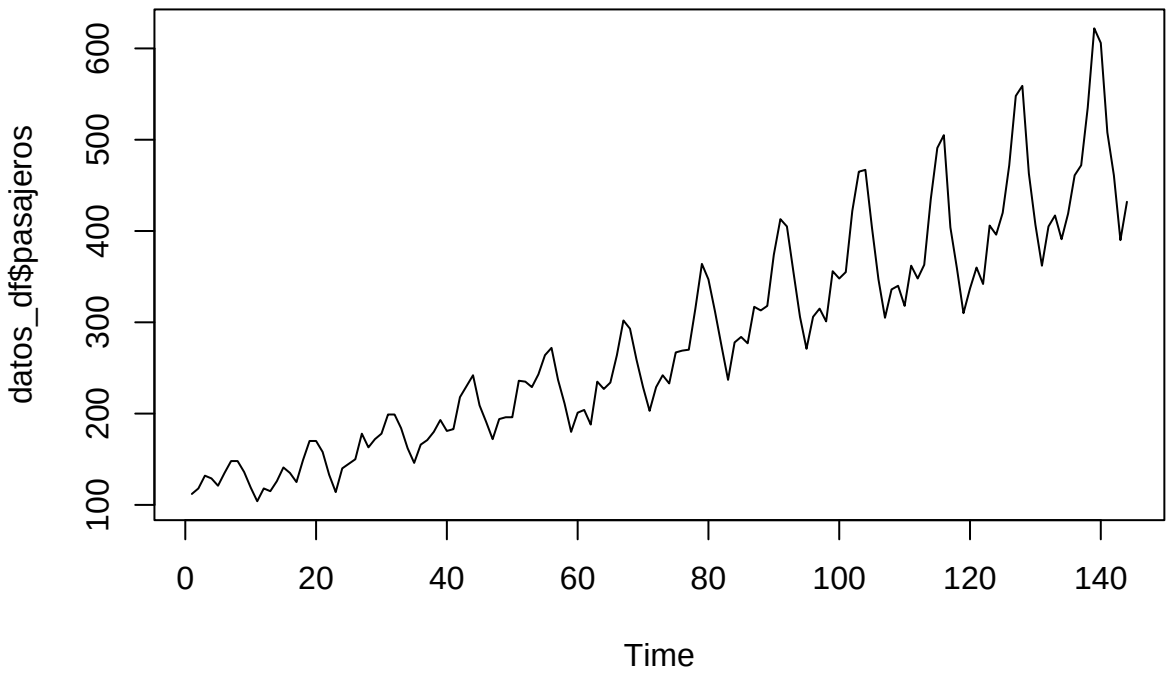
```
## [1] 1960   12
```

Las series temporales suelen representarse gráficamente con la función *ts.plot()*, tanto si se trata de objetos tipo vector o "ts". O bien con la función *plot()*.

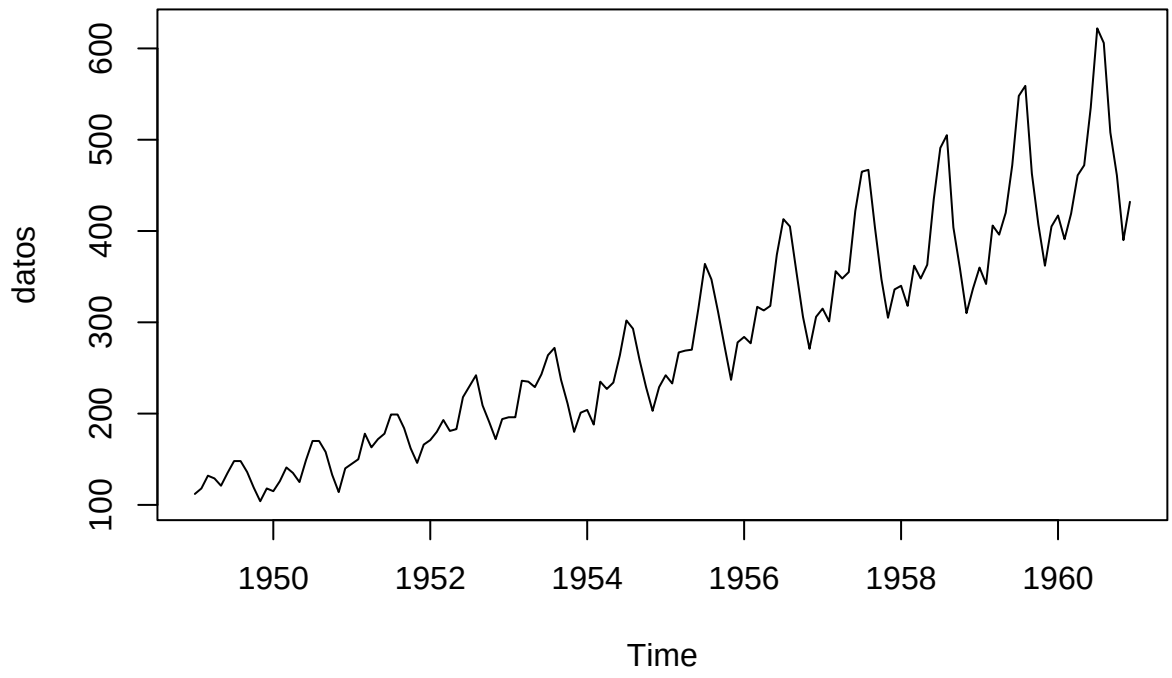
```
ts.plot(datos)
```



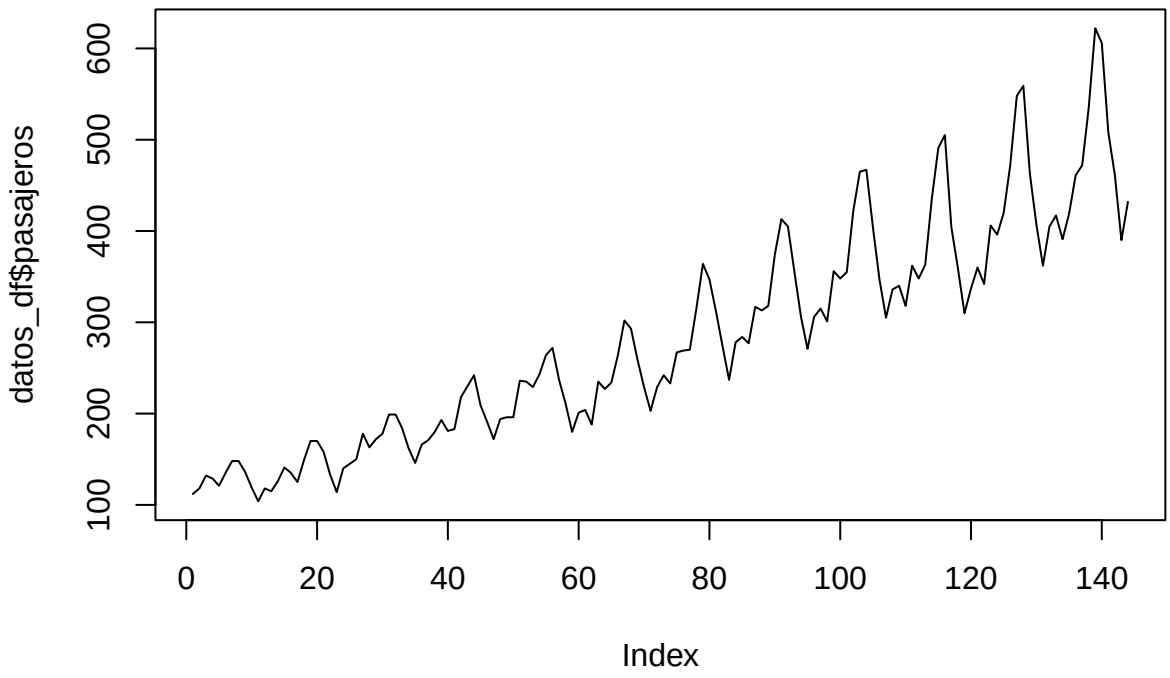

```
ts.plot(datos_df$pasajeros)
```


datos_df$pasajeros
100
200
300
400
500
600
0
20
40
60
80
100
120
140
Time

```
plot(datos)
```

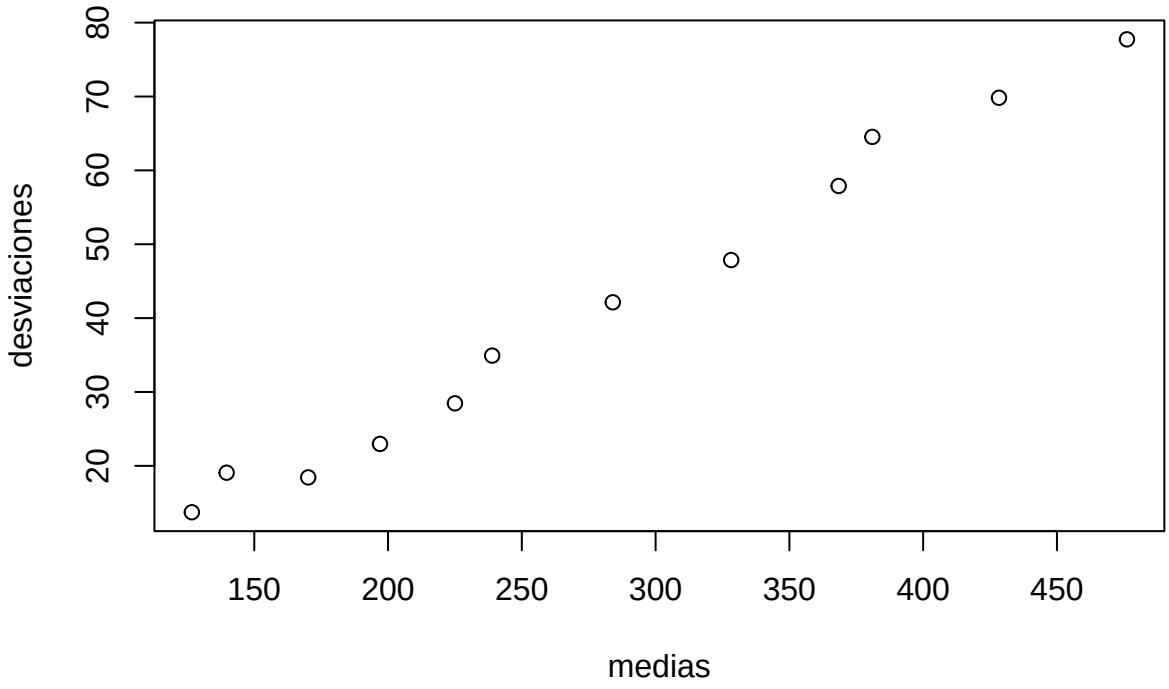



```
plot(datos_df$pasajeros, type = "l")
```



Observamos que la serie de pasajeros presenta una clara componente estacional (patrón periódico que se repite cada año), así como tendencia creciente. Además, las oscilaciones anuales se amplifican conforme aumenta el nivel de la serie, así que se trata de un esquema multiplicativo. A esta misma conclusión podemos llegar si calculamos medias y desviaciones típicas para cada año y vemos cómo la dispersión (desviación típica) aumenta conforme lo hace la media.

```
medias <- as.vector(aggregate(datos, FUN = mean))
desviaciones <- as.vector(aggregate(datos, FUN = sd))
plot(medias,desviaciones)
```



Concluimos por tanto que se trata de un esquema multiplicativo:

$x_t = T_t \times S_t \times I_t$

donde T_t representa la componente Tendencia-Ciclo, S_t la componente Estacional e I_t la componente Irregular de la serie.

En general, la representación gráfica de las series nos permitirá identificar observaciones inusuales, patrones, cambios de comportamiento en el tiempo y relaciones entre variables. Al final de esta práctica, se incluye un apartado (Apéndice) para ampliar el análisis exploratorio de series temporales.

2. Algunas operaciones con series temporales

Combinación de series temporales

Dadas dos series temporales x_t e y_t, observadas con la misma frecuencia, si queremos combinarlas en una única serie temporal multivariante (con dos columnas, una para la x_t y otra para la y_t), podemos utilizar las funciones:

ts.union(): junta las dos series, rellenando con NA los periodos en que una serie sea más larga que la otra.

intersect.ts(): junta ambas series, pero sólo con los datos correspondientes al periodo común a ambas.

```
set.seed(135)
serie1=ts(rnorm(8), freq = 4, start = c(2020,1))
serie2=ts(rnorm(10), freq = 4, start = c(2020,4))
print("Unión")
```

```
## [1] "Unión"
```

```
ts.union(serie1,serie2)
```

```
##             serie1      serie2
## 2020 Q1 -0.4450708          NA
## 2020 Q2 -0.4659652          NA
## 2020 Q3 -0.1044631          NA
## 2020 Q4  1.4083353  0.42282641
## 2021 Q1 -1.3160867  0.96486359
## 2021 Q2  0.2452088  1.00777310
## 2021 Q3  1.2047431  0.05038159
## 2021 Q4 -0.4407003  0.61003293
## 2022 Q1         NA -1.06719466
## 2022 Q2         NA  0.30162946
## 2022 Q3         NA -1.46547870
## 2022 Q4         NA -0.38005746
## 2023 Q1         NA -0.23219538
```

```
print("Intersección")
```

```
## [1] "Intersección"
```

```
ts.intersect(serie1,serie2)
```

```
##             serie1     serie2
## 2020 Q4  1.4083353 0.42282641
## 2021 Q1 -1.3160867 0.96486359
## 2021 Q2  0.2452088 1.00777310
## 2021 Q3  1.2047431 0.05038159
## 2021 Q4 -0.4407003 0.61003293
```

Extracción de un tramo temporal

La función *window()* extrae los valores de la serie temporal comprendidos entre una fecha de inicio y otra final.

```
window(serie1, start = c(2021,1),end = c(2021,4))
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2021 -1.3160867  0.2452088  1.2047431 -0.4407003
```

Serie retardada o adelantada en el tiempo

La función *lag()* construye una versión desfasada de la serie temporal, retardando (o adelantando) los valores de la serie según el parámetro k. Si no se especifica, por defecto el desfase es $k = 1$:

$lag(x_t, k) = x_{t+k}$

```
serie1
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2020 -0.4450708 -0.4659652 -0.1044631  1.4083353
## 2021 -1.3160867  0.2452088  1.2047431 -0.4407003
```

```
print("serie retardada dos instantes:")
```

```
## [1] "serie retardada dos instantes:"
```

```
lag(serie1, -2)
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2020                        -0.4450708 -0.4659652
## 2021 -0.1044631  1.4083353 -1.3160867  0.2452088
## 2022  1.2047431 -0.4407003
```

```
print("serie adelantada tres instantes:")
```

```
## [1] "serie adelantada tres instantes:"
```

```
lag(serie1, 3)
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2019            -0.4450708 -0.4659652 -0.1044631
## 2020  1.4083353 -1.3160867  0.2452088  1.2047431
## 2021 -0.4407003
```

Diferenciación de una serie

La diferenciación es una técnica utilizada habitualmente para eliminar la tendencia en una serie temporal (para conseguir que sea estacionaria).

Recordemos que dada una serie temporal, se define su **diferencia de primer orden** como la serie que resulta de restar a cada observación, la observación anterior:

$$\Delta x_t = x_t - x_{t-1}$$

Si volvemos a diferenciar la serie resultante obtenemos la serie diferencia de segundo orden. Y así podemos repetir hasta obtener la serie diferencia de orden m.

Por otra parte, las diferencias se pueden tomar con un desfase k que puede ser 1 (como hemos definido al inicio) o mayor de 1. En series temporales mensuales la diferenciación con desfase $k = 12$ suele recibir el nombre de diferenciación estacional.

$$\Delta x_{t,k} = x_t - x_{t-k}$$

A su vez, la serie resultante anterior, puede volver a diferenciarse con desfase k, obteniendo las diferencias de segundo orden y desfase k. Y en general, se puede construir la serie diferenciada de orden m y desfase k.

```
serie1
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2020 -0.4450708 -0.4659652 -0.1044631  1.4083353
## 2021 -1.3160867  0.2452088  1.2047431 -0.4407003
```

```
print("serie diferenciada de orden 1:")
```

```
## [1] "serie diferenciada de orden 1:"
```

```
diff(serie1, lag = 1, differences = 1)
```

```
##             Qtr1        Qtr2        Qtr3        Qtr4
## 2020             -0.02089445  0.36150218  1.51279839
## 2021 -2.72442205  1.56129552  0.95953435 -1.64544341
```

```
print("serie diferenciada de orden 2:")
```

```
## [1] "serie diferenciada de orden 2:"
```

```
diff(serie1, lag = 1, differences = 2)
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2020                         0.3823966  1.1512962
## 2021 -4.2372204  4.2857176 -0.6017612 -2.6049778
```

```
print("serie diferenciada de orden 1 y desfase 4:")
```

```
## [1] "serie diferenciada de orden 1 y desfase 4:"
```

```
diff(serie1, lag = 4, differences = 1)
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2021 -0.8710159  0.7111740  1.3092062 -1.8490356
```

Medias móviles de una serie

Las medias móviles eliminan las irregularidades de la serie, es decir, se produce un suavizado de la serie eliminando o atenuando el efecto de la componente irregular o incluso de la estacional, consiguiendo así identificar la trayectoria que sigue la tendencia.

Para obtener las medias móviles se puede usar la función *rollmean()* del paquete "zoo".

```
library(zoo)
```

```
##
## Adjuntando el paquete: 'zoo'
```

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

```
serie1
```

```
##            Qtr1       Qtr2       Qtr3       Qtr4
## 2020 -0.4450708 -0.4659652 -0.1044631  1.4083353
## 2021 -1.3160867  0.2452088  1.2047431 -0.4407003
```

```
print("media móvil:")
```

```
## [1] "media móvil:"
```

```
rollmean(serie1, k = 3, fill = NA, align = "center")
```

```
##              Qtr1         Qtr2         Qtr3         Qtr4
## 2020           NA -0.338499701  0.279302338 -0.004071491
## 2021  0.112485796  0.044621735  0.336417221           NA
```

3. Descomposición clásica de series temporales

3.1. Extracción de las componentes

La descomposición clásica de una serie temporal se puede realizar con la función *decompose()* de R. Es imprescindible que los datos en estudio se correspondan con un objeto de clase "ts" para que R reconozca el periodo de la componente estacional (el argumento "frequency"). Además, debemos indicar si las componentes se combinan con un esquema aditivo o multiplicativo, lo cual podemos decidir con anterioridad a partir de la representación gráfica. Como resultado, obtendremos tres series: la componente tendencia (trend), la componente estacional (seasonal) y la componente irregular (random).

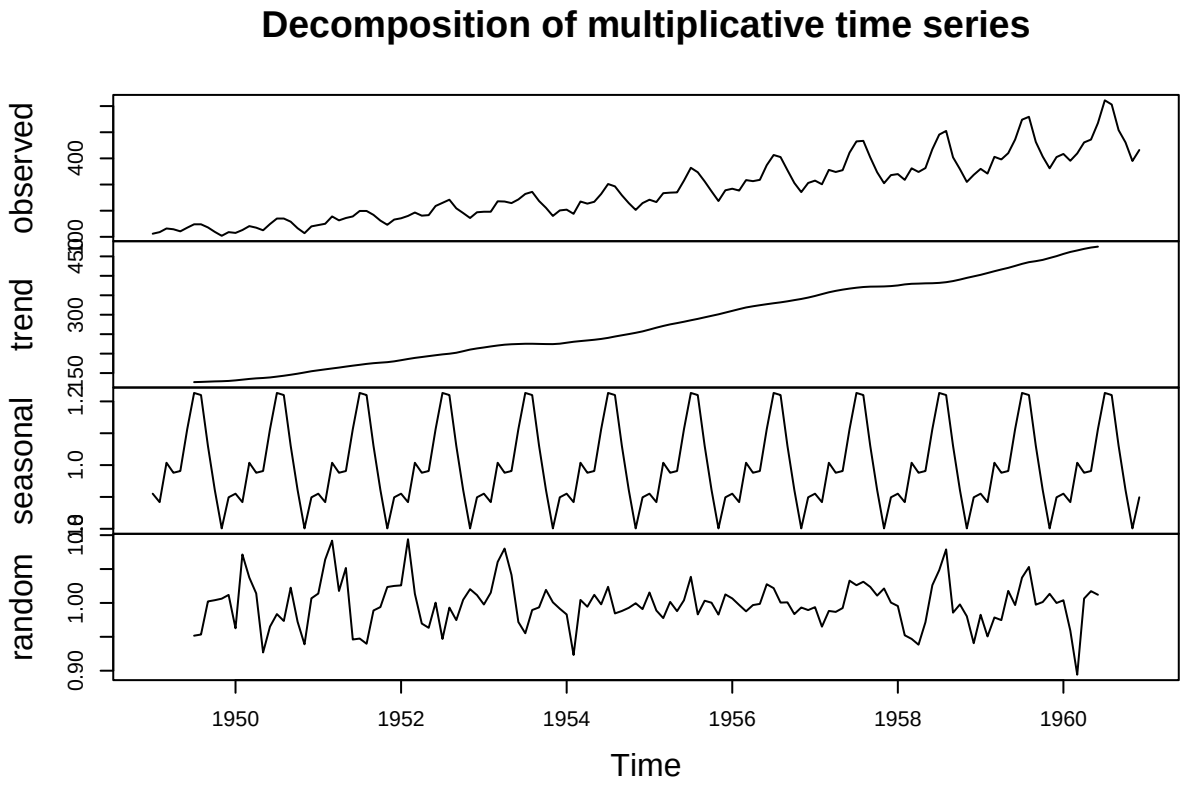
```
descomposicion_clasica <- decompose(datos, type = "multiplicative")
#descomposicion_clasica2 <- decompose(datos_ts, type = "multiplicative")
```

```
# str(descomposicion_clasica)
summary(descomposicion_clasica)
```

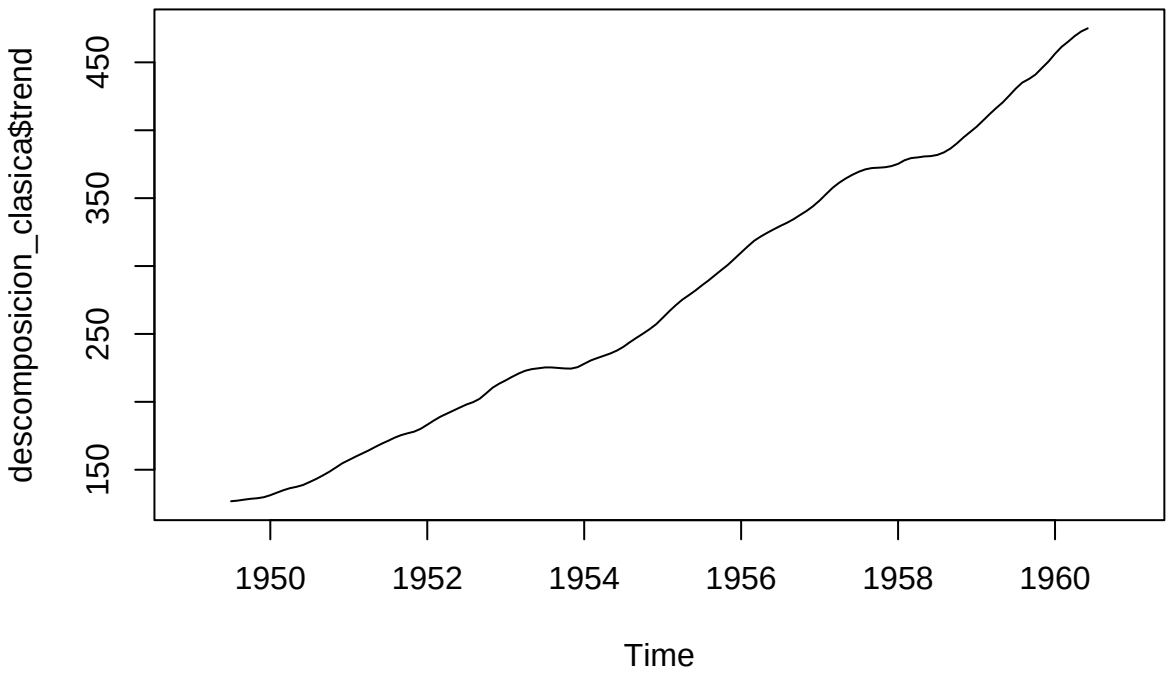
```
##          Length Class  Mode
## x        144    ts     numeric
## seasonal 144    ts     numeric
## trend    144    ts     numeric
## random   144    ts     numeric
## figure    12    -none- numeric
## type       1    -none- character
```

Con la función *plot()* aplicada a la salida de la descomposición, podemos representar conjuntamente todas las componentes resultantes. También podemos representarlas cada una por separado usando *ts.plot()* aplicada a cada componente de tipo "ts". Así podremos observar con más claridad que la tendencia sigue un patrón aproximadamente lineal.

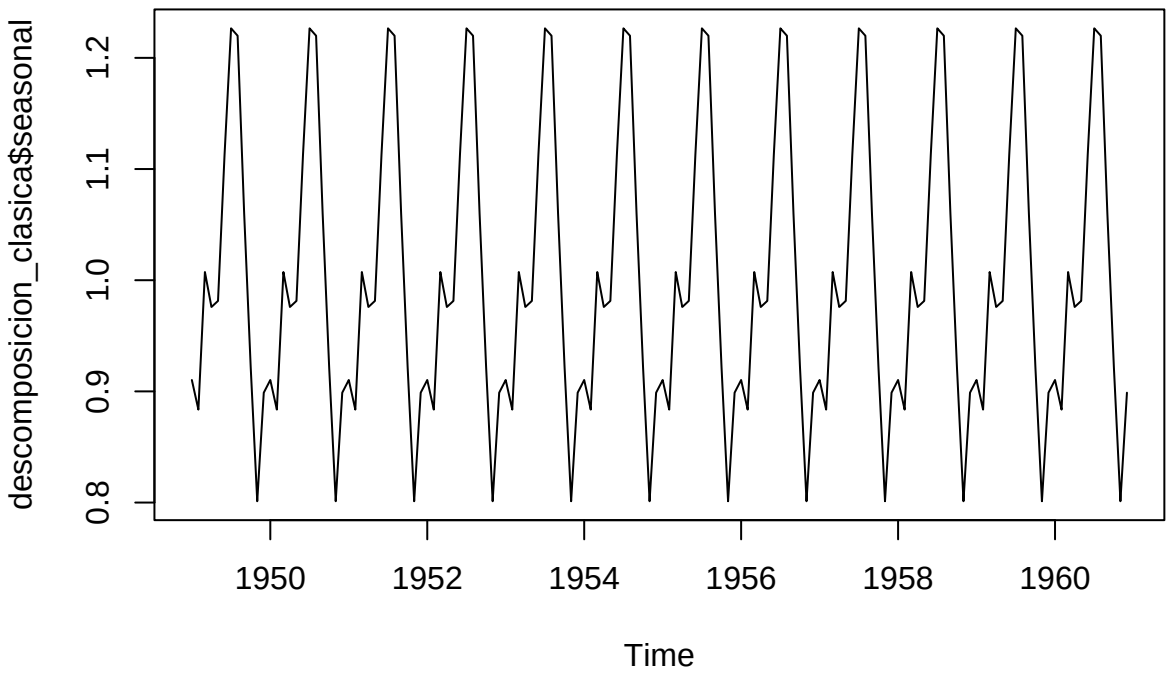
```
plot(descomposicion_clasica)
```



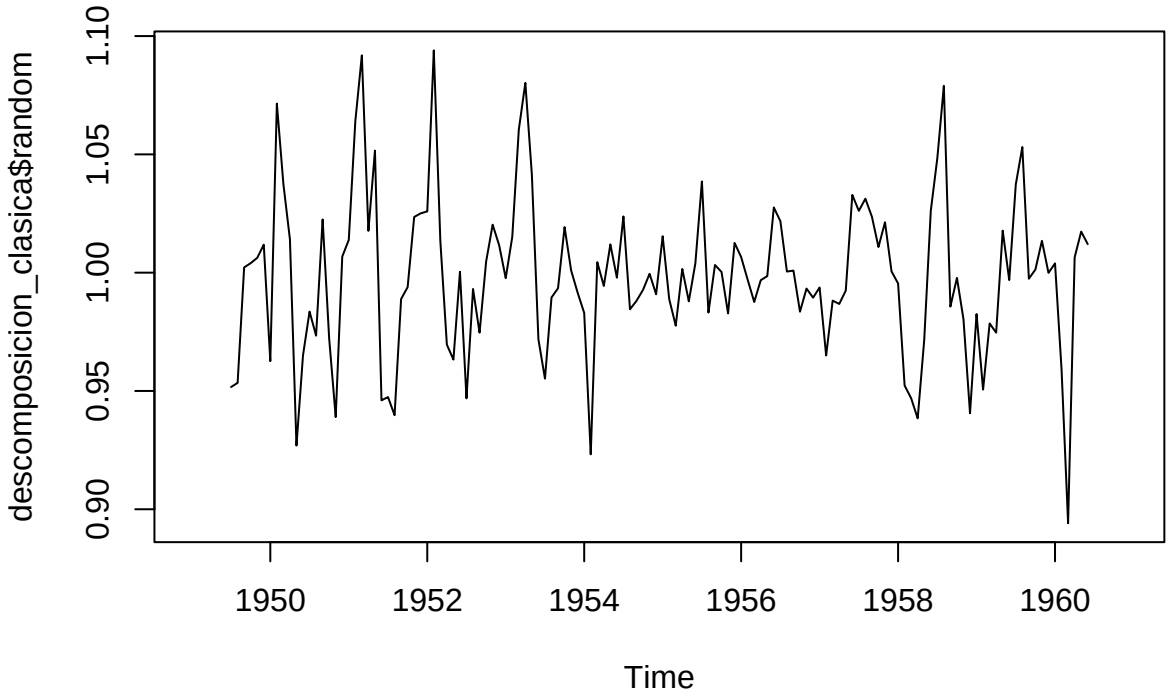

```
ts.plot(descomposicion_clasica$trend)
```




```
ts.plot(descomposicion_clasica$seasonal)
```




```
ts.plot(descomposicion_clasica$random)
```



3.2. Predicciones: obtención de un modelo determinista

En ocasiones resulta necesario obtener un modelo determinista (una expresión formal) que permita explicar el comportamiento de la serie observada con el fin de realizar predicciones. Para ello, tendremos en cuenta que:

a) La estimación de la componente Estacional (S_t) para predicciones futuras viene dada Índices de Variación Estacional obtenidos en la descomposición, pues se supone que la estacionalidad permanece estable en el tiempo.

b) La estimación de la componente Irregular (I_t) para observaciones futuras se supone constante e igual a 1 por ser esquema multiplicativo (si fuera un esquema aditivo, se supondría constante e igual a cero).

Por tanto, sólo nos faltaría obtener una expresión formal para representar la evolución de la tendencia y así poder realizar predicciones para instantes futuros. Ajustaremos la tendencia por mínimos cuadrados en función del tiempo.

```
tiempo <- 1:length(datos)
tendencia.lm <- lm(descomposicion_clasica$trend ~ tiempo)
tendencia.lm
```

```
##
## Call:
## lm(formula = descomposicion_clasica$trend ~ tiempo)
##
## Coefficients:
## (Intercept)       tiempo
##      84.648        2.667
```

Por tanto, la componente Tendencia-Ciclo puede expresarse en función del tiempo:

$T_t = 84.648 + 2.667 \times t$

donde t representa el tiempo. Entonces, la parte determinista de la serie temporal que sirve para realizar predicciones se puede modelizar mediante:

$\hat{x_t} = \hat{T_t} \times \hat{S_t} = (84.648 + 2.667 \times t) \times \hat{S_t}$

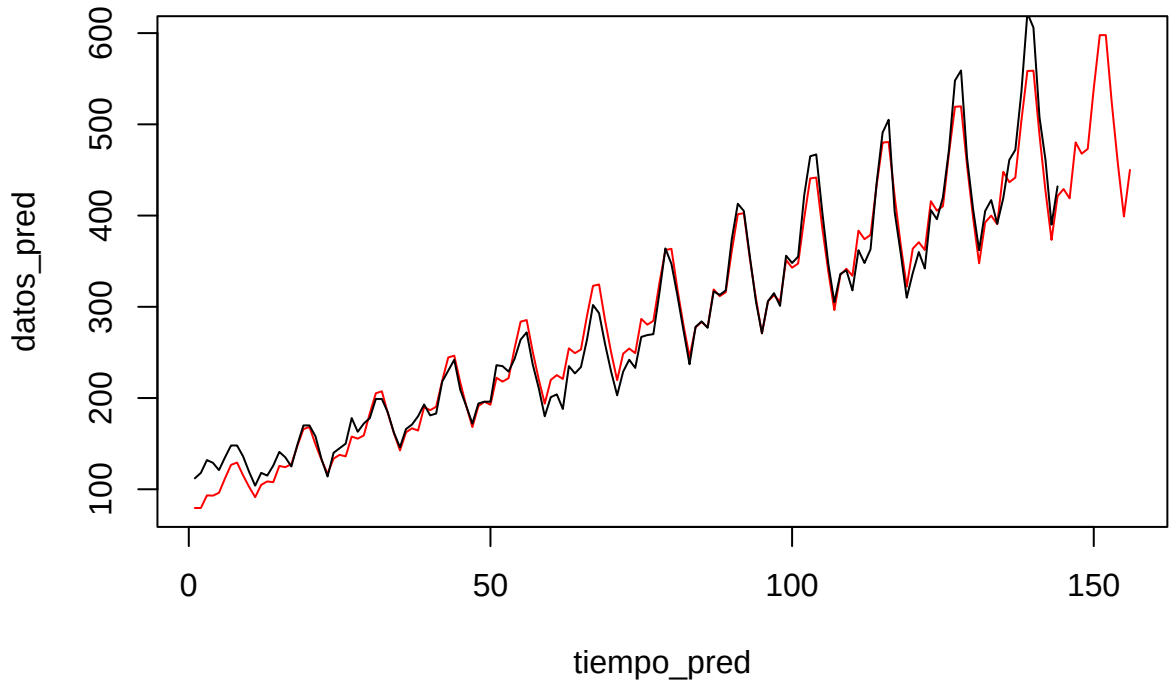
¿Cómo obtener los valores ajustados en el tramo observado y realizar predicciones en instantes futuros?

Veamos cómo calcular los valores ajustados y las predicciones para los 12 meses del año 1961. Comenzamos definiendo el tramo de tiempos de interés (en nuestro caso, los años observados y el año siguiente), así como la predicción de Tendencia y Estacionalidad en dicho tramo:

```
tiempo_pred <- 1:(length(datos) + 12)

estacional_pred <- c(descomposicion_clasica$seasonal,
                     descomposicion_clasica$seasonal[1:12])

tendencia_pred <- predict(tendencia.lm, data.frame(tiempo = tiempo_pred))
```

Las predicciones se obtienen de la siguiente forma, donde además representamos la serie original de pasajeros (en negro) y la serie predicha (en rojo).

```
datos_pred <- tendencia_pred * estacional_pred
plot(tiempo_pred, datos_pred, type="l", col = "red")
lines(tiempo,datos_df$pasajeros,type="l")
```



Aunque obtenemos un ajuste relativamente bueno, podemos observar que la tendencia de la serie original presenta cierta curvatura respecto a las predicciones obtenidas. Podríamos intentar proponer un modelo

cuadrático o exponencial para explicar la tendencia en función del tiempo.

Residuos y medidas de error (bondad del ajuste)

Calculamos los residuos y medidas de error asociadas de la siguiente forma:

```
residuos <- datos - datos_pred[1:length(datos)]
MAE <- mean(abs(residuos))
RMSE <- sqrt(mean(residuos^2))
MAE
```

```
## [1] 13.45225
```

```
RMSE
```

```
## [1] 17.32319
```

Recordemos que otra forma de evaluar la adecuación del modelo propuesto consiste en comporarlo con métodos naive (ingenuos).

4. Descomposición STL de series temporales

4.1. Extracción de las componentes con STL

En este apartado veremos cómo descomponer la misma serie de pasajeros, pero ahora usando la descomposición STL. Destacar que dicha descomposición sólo puede aplicarse sobre series que se combinan siguiendo **esquema aditivo**. Como en nuestro caso el esquema es multiplicativo, debemos tomar logaritmos neperianos sobre la serie y así transformarlo en un esquema aditivo:

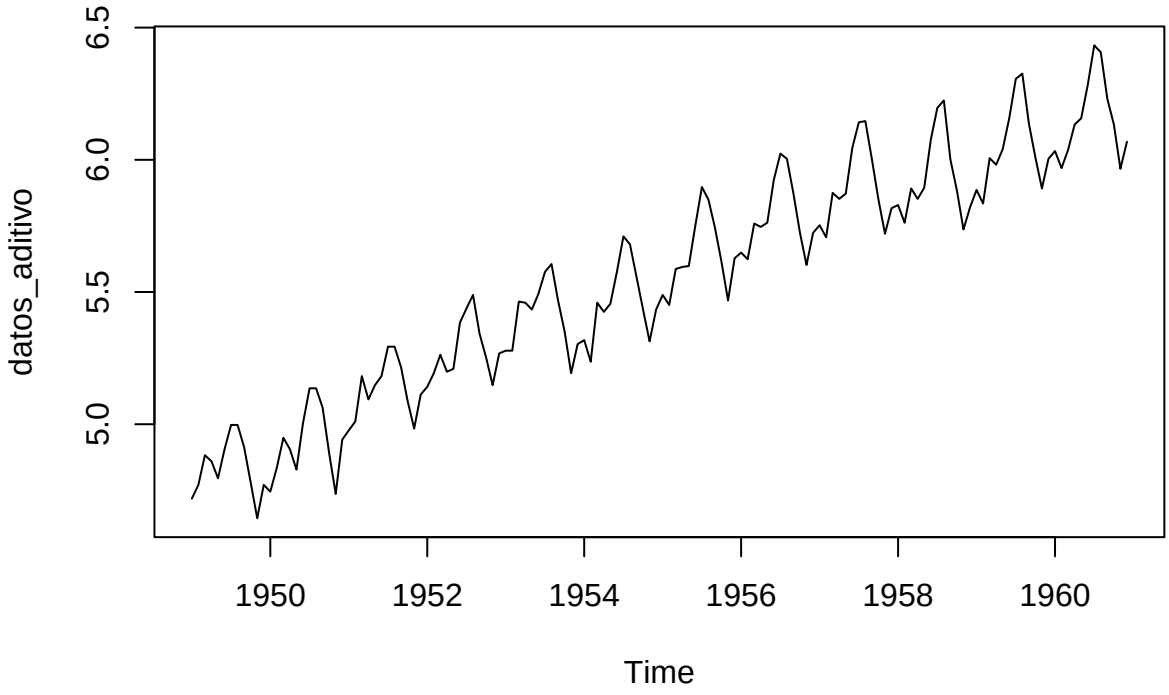
$x_t = T_t \times S_t \times I_t \implies log(x_t) = log(T_t) + log(S_t) + log(I_t)$

Luego:

$log(x_t) = \widetilde{T}_t + \widetilde{S}_t + \widetilde{I}_t \implies x_t = \exp(\widetilde{T}_t) \times \exp(\widetilde{S}_t) \times \exp(\widetilde{I}_t)$

donde $\widetilde{T}_t = log(T_t), \widetilde{S}_t = log(S_t), \widetilde{I}_t = log(I_t)$

```
datos_aditivo <- log(AirPassengers)
ts.plot(datos_aditivo)
```



Al tomar logaritmos sobre la serie de pasajeros, observamos que las fluctuaciones anuales de la serie mantienen aproximadamente su dispersión aunque aumente la media, tratándose por tanto de un esquema aditivo. Por otra parte, adevertimos una clara componente estacional (patrón periódico que se repite cada año), pero que varía su forma con el tiempo. Por este motivo, la descomposición STL será más apropiada que la clásica, ya que permite un patrón periódico flexible frente al rígido del enfoque clásico.

Veamos cómo obtener las componentes aditivas $(\widetilde{T}_t, \widetilde{S}_t, \widetilde{I}_t)$ usando el método STL:

```
STL <- stl(datos_aditivo, s.window = 7)
#str(STL)
summary(STL)
```

```
##  Call:
##  stl(x = datos_aditivo, s.window = 7)
##
##  Time.series components:
##     seasonal              trend            remainder
##  Min.   :-0.22188182   Min.   :4.809113   Min.   :-0.06875615
##  1st Qu.:-0.08857735   1st Qu.:5.201819   1st Qu.:-0.01027597
##  Median :-0.01380803   Median :5.552237   Median :-0.00104014
##  Mean   : 0.00053801   Mean   :5.541836   Mean   :-0.00019840
##  3rd Qu.: 0.07274467   3rd Qu.:5.916865   3rd Qu.: 0.01350479
##  Max.   : 0.25725445   Max.   :6.197002   Max.   : 0.06572901
##  IQR:
##      STL.seasonal STL.trend STL.remainder data
##      0.16132      0.71505   0.02378       0.69453
##    %  23.2        103.0       3.4         100.0
```

```
##
##  Weights: all == 1
##
##  Other components: List of 5
##  $ win  : Named num [1:3] 7 23 13
##  $ deg  : Named int [1:3] 0 1 1
##  $ jump : Named num [1:3] 1 3 2
##  $ inner: int 2
##  $ outer: int 0
```

El argumento "s.window" indica el ancho de banda usado para el suavizado LOESS de la estacionalidad. Valores pequeños permiten cambios más rápidos en la componente estacional. Y en el caso de usar todos los periodos de la serie se considera estacionalidad estable (igual que en enfoque clásico).

Al igual que en la descomposición clásica, el método STL devuelve tres series en el siguiente orden:la componente estacional (seasonal), la componente tendencia (trend) y la componente irregular (reminder).

```
estacional_STL <- STL$time.series[, 1]
tendencia_STL <- STL$time.series[, 2]
irregular_STL <- STL$time.series[ , 3]

# componentes_STL <- STL[["time.series"]]
# estacional_STL <- componentes_STL[, 1]
# tendencia_STL <- componentes_STL[, 2]
# irregular_STL <- componentes_STL[ , 3]
```

Otra forma de extraer las componentes resultantes de la descomposición STL es usando el paquete *forecast*.

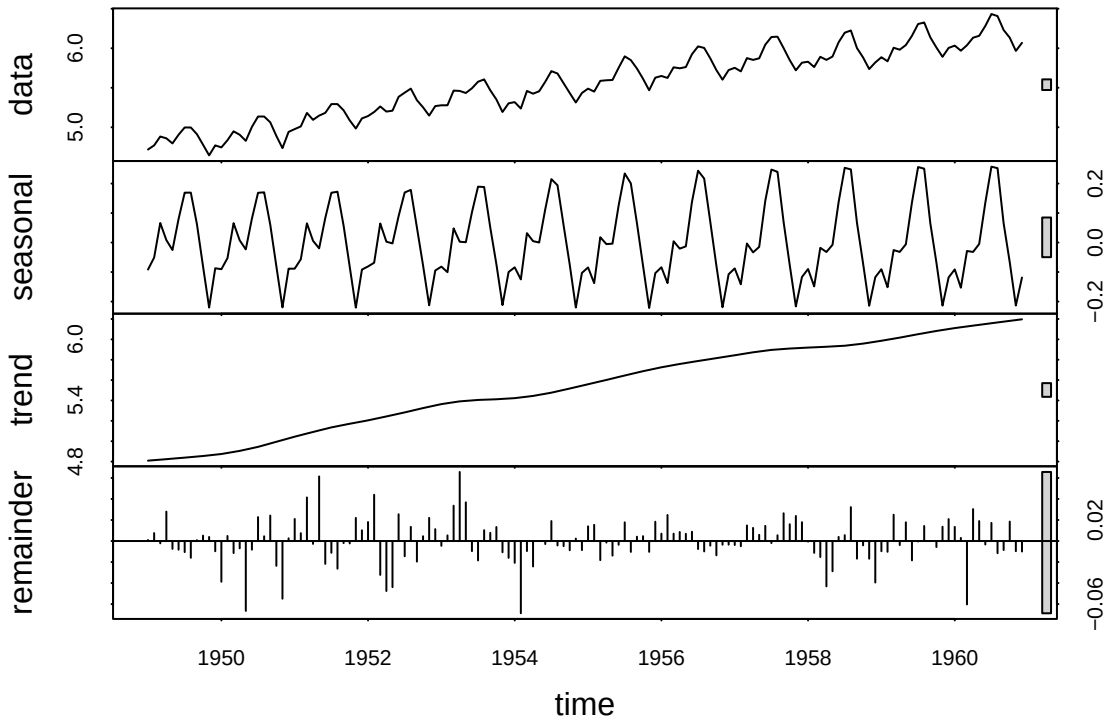
```
library(forecast)
```

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

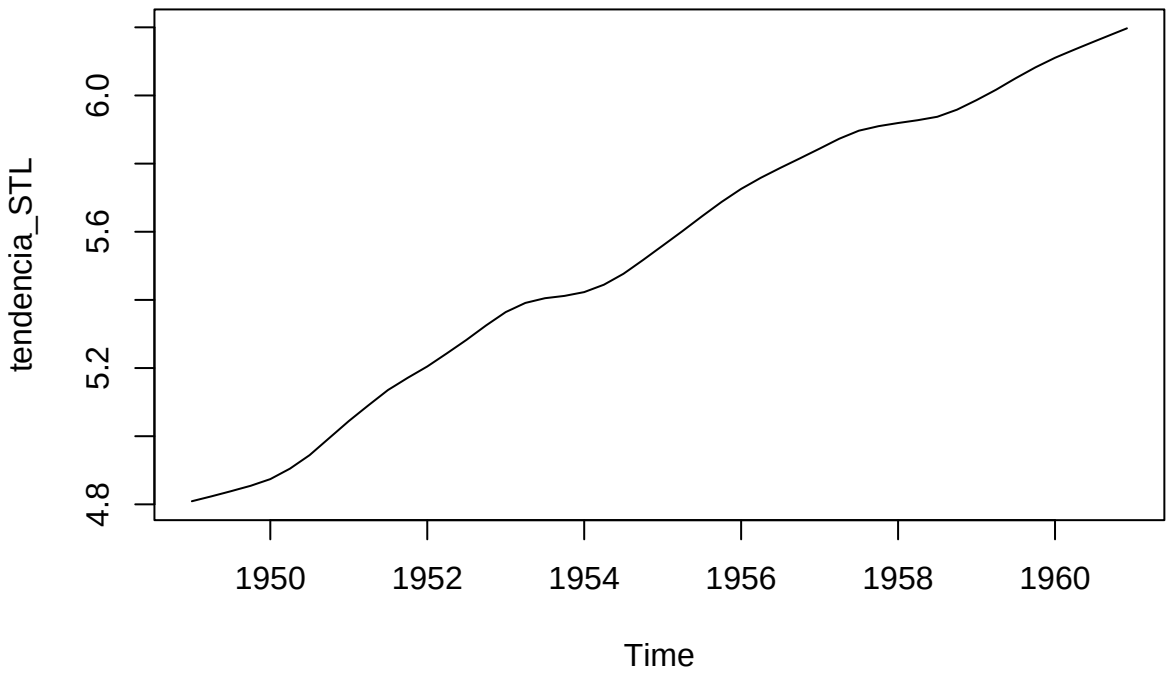
```
Estacional_STL <- seasonal(STL)
Tendencia_STL <- trendcycle(STL)
Irregular_STL <- remainder(STL)
```

Con la función *plot()* aplicada a la salida de la descomposición, podemos representar conjuntamente todas las componentes resultantes. También podemos representarlas cada una por separado usando *ts.plot()* aplicada a cada componente de tipo "ts". Así podremos observar con más claridad que la tendencia sigue un patrón aproximadamente lineal. También que la componente estacional no es estable sino que varía con el tiempo, y que su media anual es cero (normalización).

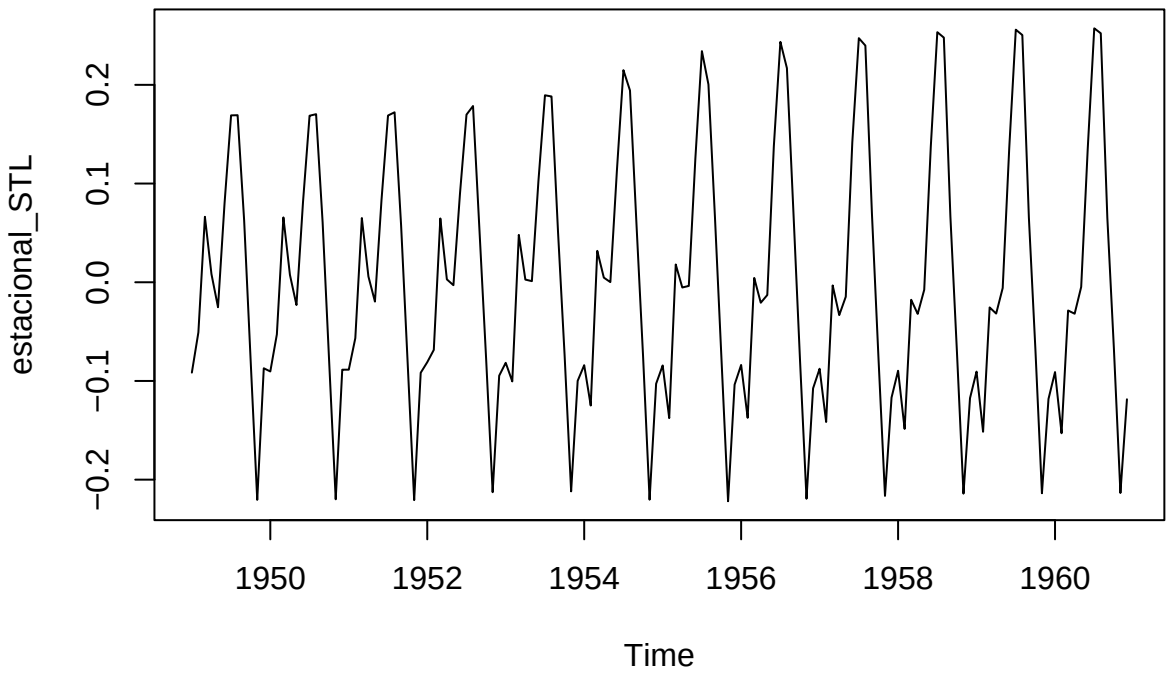
```
plot(STL)
```



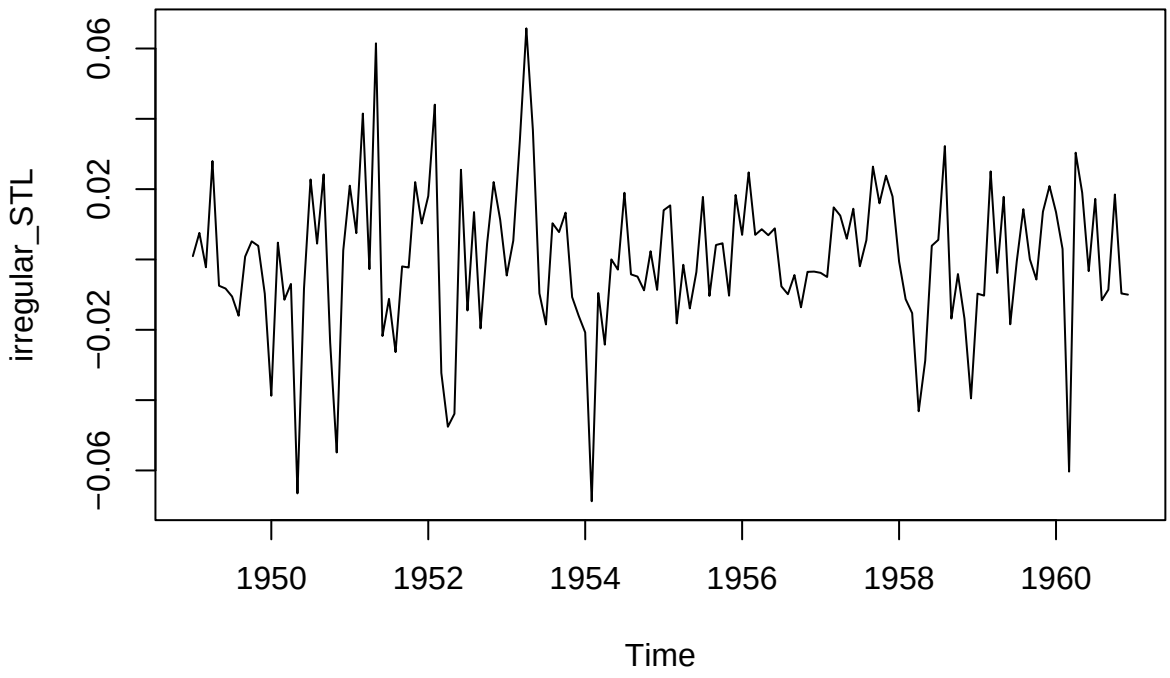

```
# plot(STL$time.series)

ts.plot(tendencia_STL)
```




```
ts.plot(estacional_STL)
```



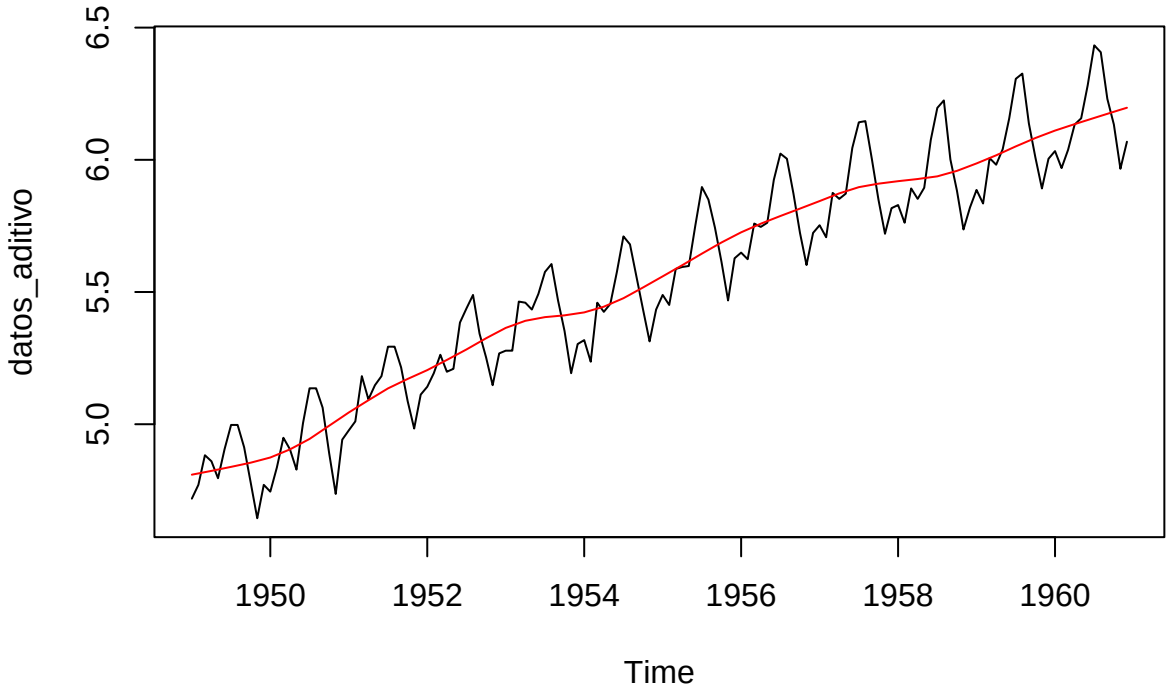

```
ts.plot(irregular_STL)
```




```
# ts.plot(Tendencia_STL)
# ts.plot(Estacional_STL)
# ts.plot(Irregular_STL)
```

Representemos ahora la serie de datos aditiva junto con la tendencia extraída por STL:

```
plot(datos_aditivo)
lines(tendencia_STL, col = "red")
```

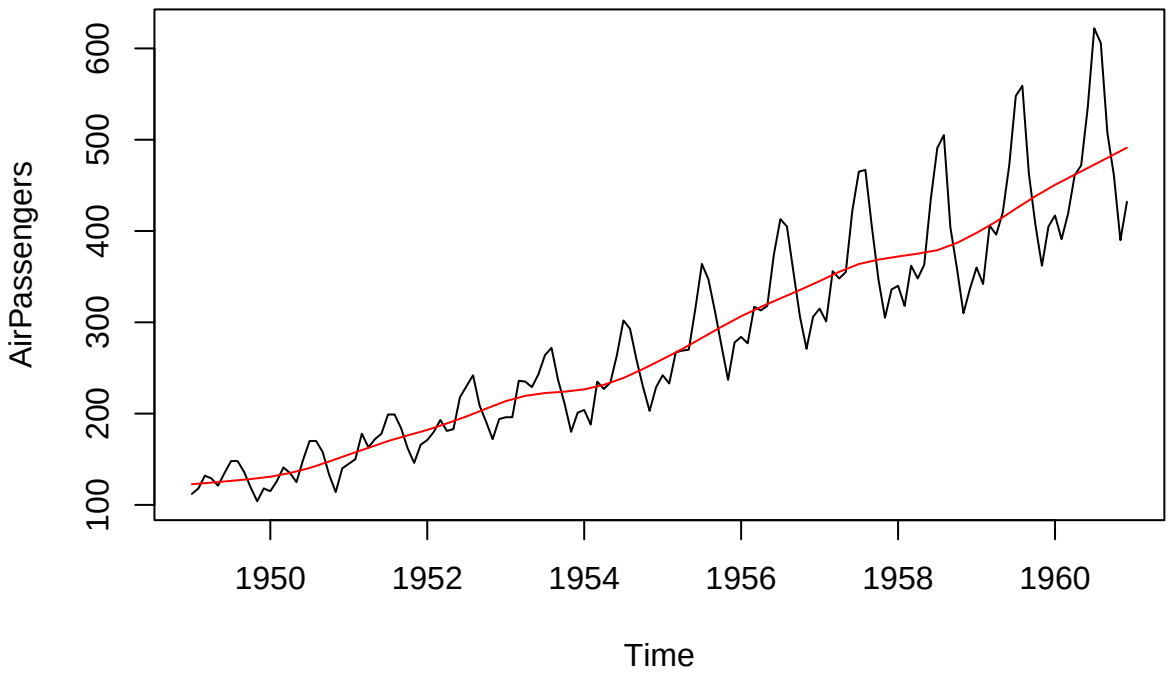


Recordemos que el método STL ha extraido las componentes del esquema aditivo. Si queremos las componentes de la serie original (esquema multiplicativo), tendremos que deshacer el cambio tomando exponenciales.

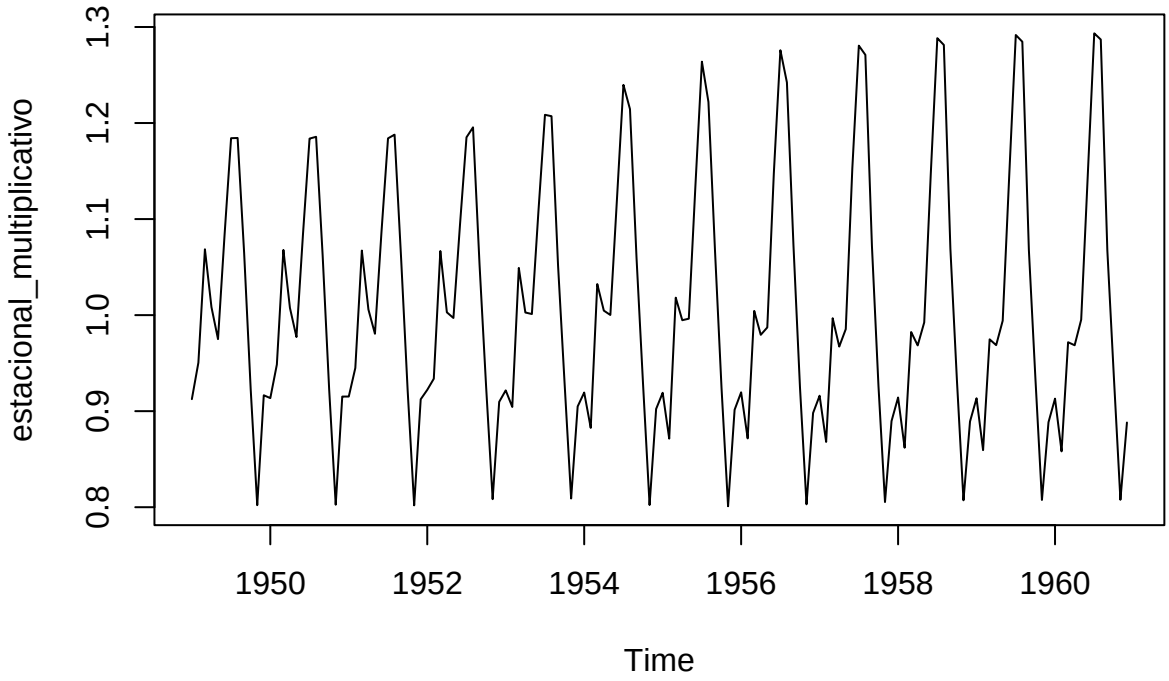
```
estacional_multiplicativo <- exp(estacional_STL)
tendencia_multiplicativo <- exp(tendencia_STL)
irregular_multiplicativo <- exp(irregular_STL)
```

Representemos la serie original (esquema multiplicativo) y sus componentes. Para el esquema multiplicativo, estacionalidad y componente irregular oscilan alrededor del 1 en lugar del cero.

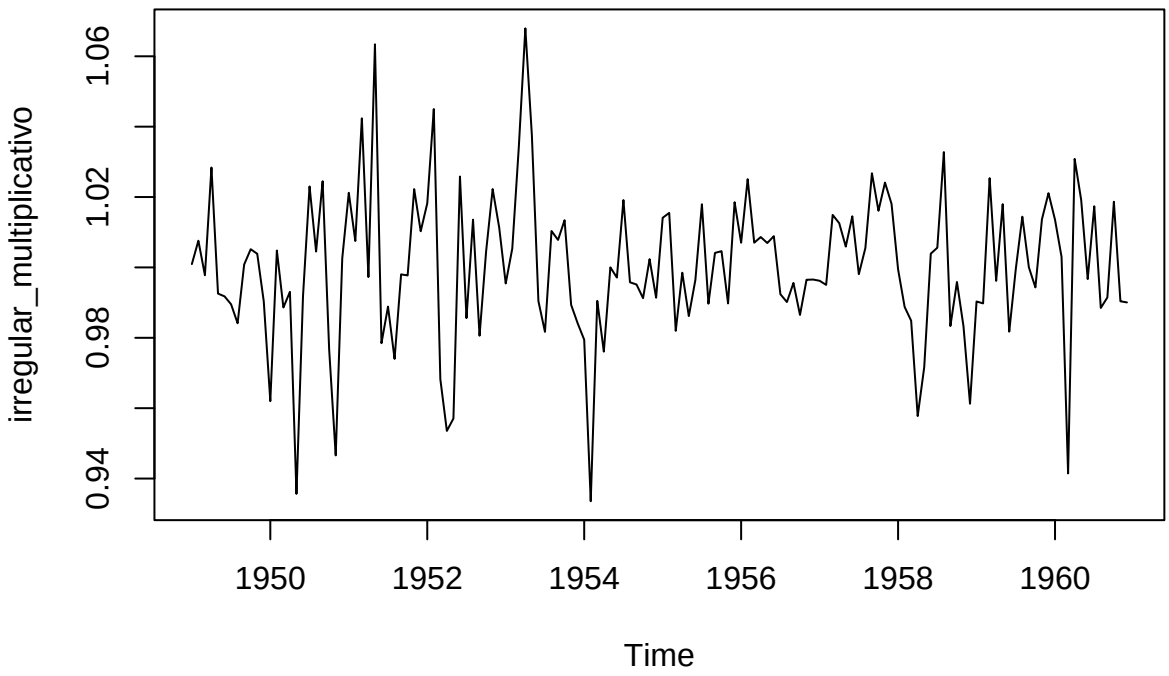
```
plot(AirPassengers)
lines(tendencia_multiplicativo, col = "red")
```




```
ts.plot(estacional_multiplicativo)
```




```
ts.plot(irregular_multiplicativo)
```



4.2. Predicciones con descomposición STL

Para realizar predicciones con STL tendremos en cuenta lo siguiente:

a) La estimación de la componente Irregular aditiva ($\widetilde{I}_t$) para observaciones futuras se supone constante e igual a 0.

b) La estimación de la componente Estacional aditiva ($\widetilde{S}_t$) para predicciones futuras vendrá dada por los valores de la componente estacional del último año observado. Recordemos que para STL la estacionalidad varía con el tiempo.

c) Para poder realizar predicciones de la tendencia en instantes futuros, ajustaremos la componente Tendencia aditiva ($\widetilde{T}_t$) por mínimos cuadrados en función del tiempo.

```
tiempo2 <- 1:length(datos_aditivo)
tendencia_aditivo.lm <- lm(tendencia_STL ~ tiempo2)
tendencia_aditivo.lm
```

```
##
## Call:
## lm(formula = tendencia_STL ~ tiempo2)
##
## Coefficients:
## (Intercept)      tiempo2
##     4.81357      0.01005
```

Por tanto, la componente Tendencia-Ciclo aditiva puede expresarse en función del tiempo:

$\widetilde{T}_t = 4.81357 + 0.01005 \times t$

donde t representa el tiempo. Entonces, la parte determinista de la serie temporal que sirve para realizar predicciones se puede modelizar mediante:

$\widehat{log(x_t)} = (4.81357 + 0.01005 \times t) + \widetilde{S}_t$

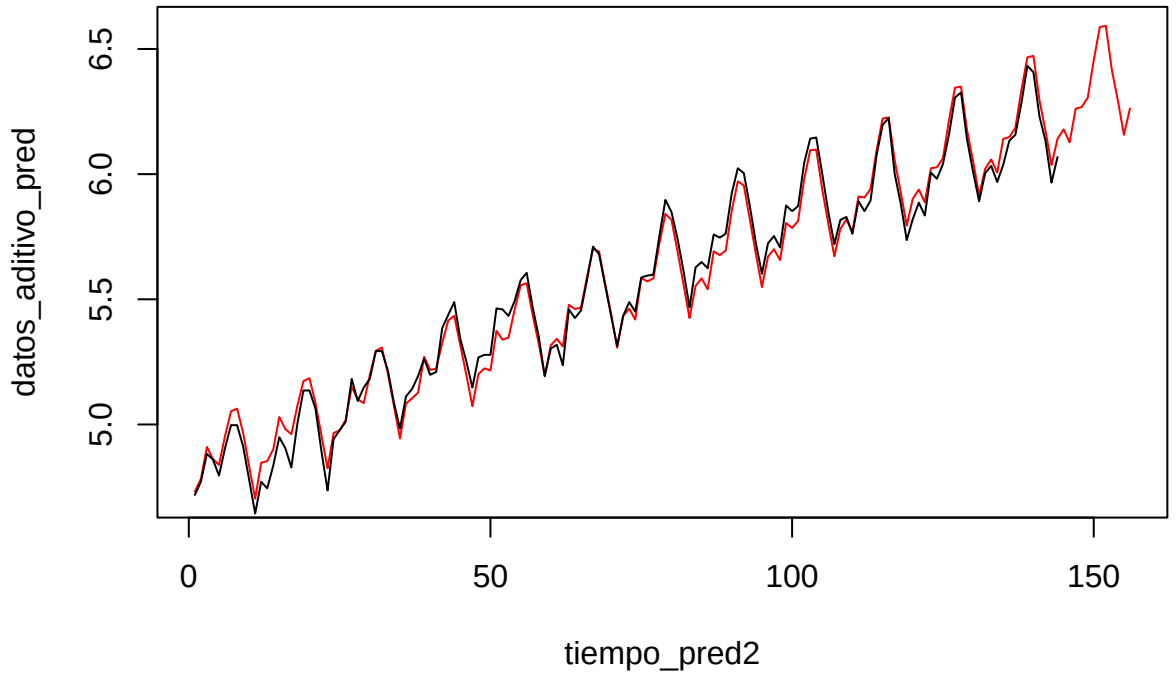
¿Cómo obtener los valores ajustados en el tramo observado y realizar predicciones en instantes futuros?

Veamos cómo calcular los valores ajustados y las predicciones para los 12 meses del año 1961. Comenzamos definiendo el tramo de tiempos de interés (en nuestro caso, los años observados y el año siguiente), así como la predicción de Tendencia y Estacionalidad en dicho tramo:

```
tiempo_pred2 <- 1:(length(datos_aditivo) + 12)

estacional_STL_pred <- c(estacional_STL, estacional_STL[133:144])

tendencia_STL_pred <- predict(tendencia_aditivo.lm,
                              data.frame(tiempo2 = tiempo_pred2))
```

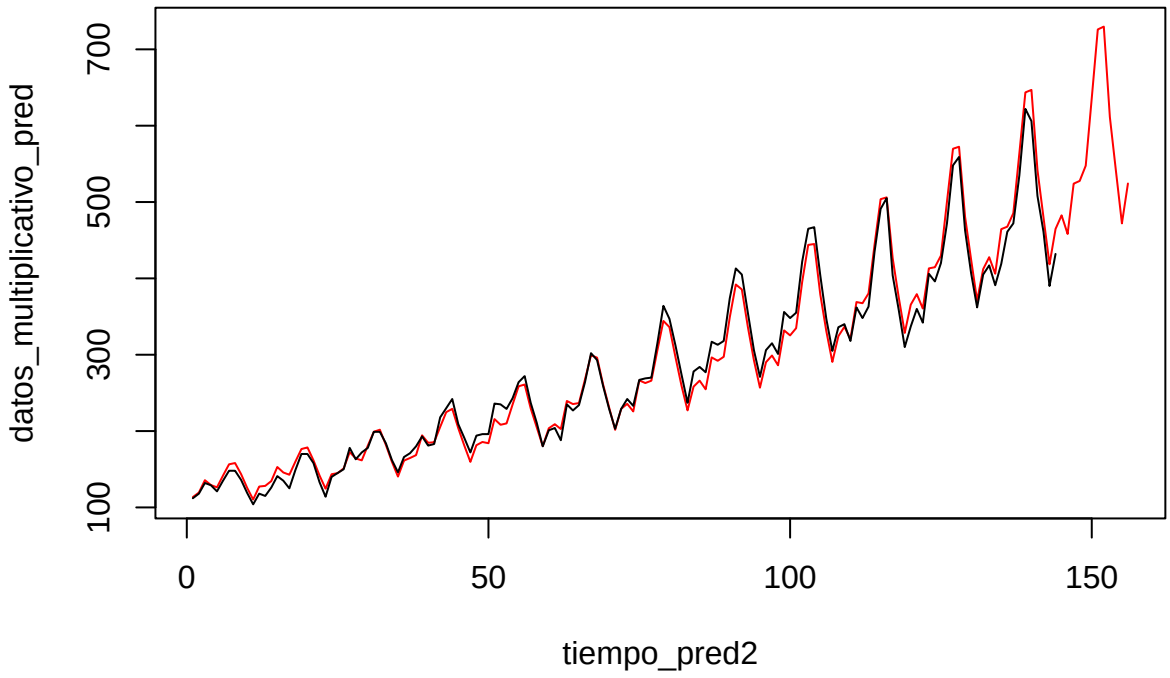
Las predicciones del esquema aditivo se obtienen de la siguiente forma, donde además representamos la serie aditiva (en negro) y su predicción (en rojo).

```
datos_aditivo_pred <- tendencia_STL_pred + estacional_STL_pred
plot(tiempo_pred2, datos_aditivo_pred, type="l", col = "red")
lines(tiempo2, datos_aditivo, type="l")
```



Por tanto, las predicciones de la serie original de pasajeros (esquema multpilicativo) se obtienen tomando exponenciales.

```
datos_multiplicativo_pred <- exp(datos_aditivo_pred)
plot(tiempo_pred2, datos_multiplicativo_pred, type="l", col = "red")
lines(tiempo2, AirPassengers, type="l")
```



Observamos un ajuste bastante bueno, mejor que en el caso de la descomposición clásica.

Residuos y medidas de error (bondad del ajuste)

Calculamos los residuos de la serie original (esquema multiplicativo) y medidas de error asociadas de la siguiente forma:

```
residuos_multiplicativo <- AirPassengers - datos_multiplicativo_pred[1:length(AirPassengers)]
MAE_multiplicativo <- mean(abs(residuos_multiplicativo))
RMSE_multiplicativo <- sqrt(mean(residuos_multiplicativo^2))
MAE_multiplicativo
```

```
## [1] 12.00082
```

```
RMSE_multiplicativo
```

```
## [1] 14.85867
```

Podemos comprobar que tanto el MAE como el RMSE son menores usando la descomposición STL frente a la descomposición clásica.

NOTA: Indicar que el paquete forecast permite realizar predicciones automáticas usando la descomposición STL. En realidad, predice la serie desestacionalizada con métodos que veremos en temas posteriores y posteriormente le añade la componente estacional del último año observado.

```
# library(forecast)
# forecast(STL)
# stlf(log(datos), s.window = 7)
# stlf(log(datos)) #Esta opcion tiene un s.window automatico
```

5. APÉNDICE: Gráficos estacionales y de subseries

En un análisis exploratorio de series temporales conviene contemplar diferentes representaciones gráficas de las series en estudio. Aunque podrían realizarse con funciones base de R, resulta más tedioso que si usamos funciones ya preparadas para ese fin.

En este apartado veremos cómo representar gráficos estacionales y de subseries usando funciones del paquete "fpp3". Los ejemplos de este apartado se han extraído del libro online https://otexts.com/fpp3/ .

Por ejemplo, usaremos los datos de demanda eléctrica (cada 30 minutos) en Victoria (Australia), desde el inicio de 2012 hasta el final de 2014. Dichos datos están disponibles en el conjunto de datos *vic_elec*, donde también se encuentran las fechas, un indicador de festividad y las temperaturas.

```
library(fpp3)
```

```
## Registered S3 method overwritten by 'tsibble':
##   method               from
##   as_tibble.grouped_df dplyr
```

```
## -- Attaching packages --------------------------------------------- fpp3 1.0.1 --
```

```
## v tibble      3.2.1     v tsibble     1.1.5
## v dplyr       1.1.4     v tsibbledata 0.4.1
## v tidyr       1.3.1     v feasts      0.4.1
## v lubridate   1.9.3     v fable       0.4.0
## v ggplot2     3.5.1
```

```
## -- Conflicts ------------------------------------------------ fpp3_conflicts --
## x lubridate::date()    masks base::date()
## x dplyr::filter()      masks stats::filter()
## x tsibble::index()     masks zoo::index()
## x tsibble::intersect() masks base::intersect()
## x tsibble::interval()  masks lubridate::interval()
## x dplyr::lag()         masks stats::lag()
## x tsibble::setdiff()   masks base::setdiff()
## x tsibble::union()     masks base::union()
```

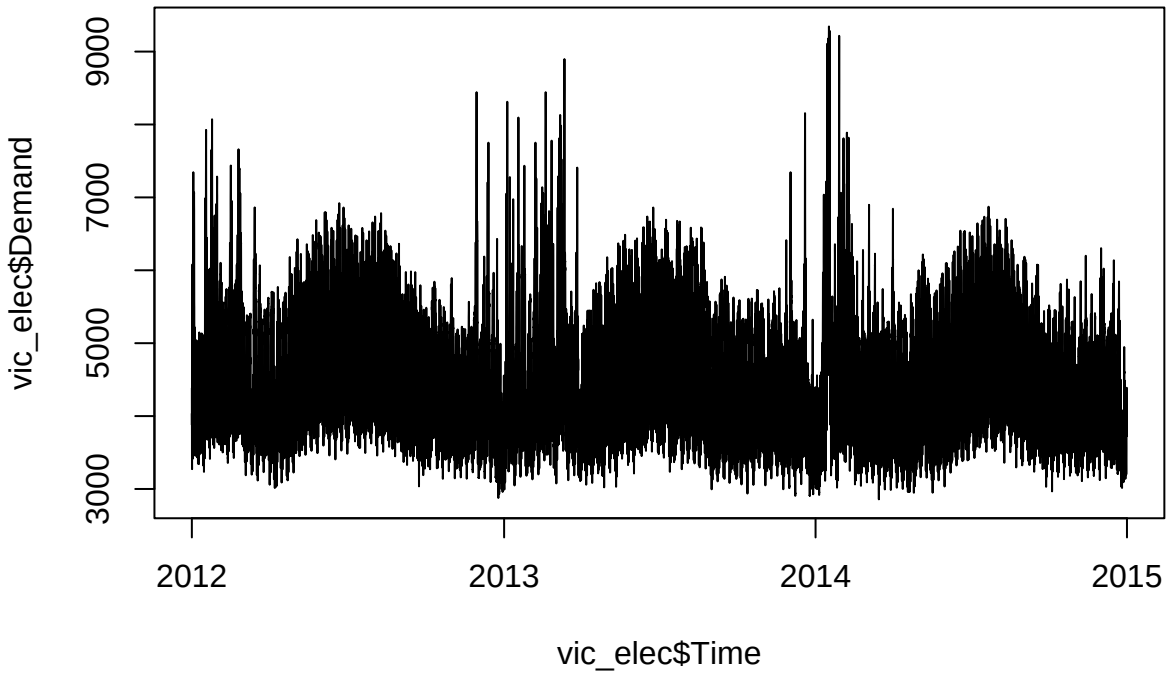
```
# str(vic_elec)
summary(vic_elec)
```

```
##       Time                          Demand      Temperature
##  Min.   :2012-01-01 00:00:00   Min.   :2858   Min.   : 1.50
##  1st Qu.:2012-09-30 22:52:30   1st Qu.:3969   1st Qu.:12.30
##  Median :2013-07-01 22:45:00   Median :4635   Median :15.40
##  Mean   :2013-07-01 22:45:00   Mean   :4665   Mean   :16.27
##  3rd Qu.:2014-04-01 23:37:30   3rd Qu.:5244   3rd Qu.:19.40
##  Max.   :2014-12-31 23:30:00   Max.   :9345   Max.   :43.20
##       Date              Holiday
##  Min.   :2012-01-01   Mode :logical
##  1st Qu.:2012-09-30   FALSE:51120
##  Median :2013-07-01   TRUE :1488
##  Mean   :2013-07-01
```

```
##  3rd Qu.:2014-04-01
##  Max.   :2014-12-31
```

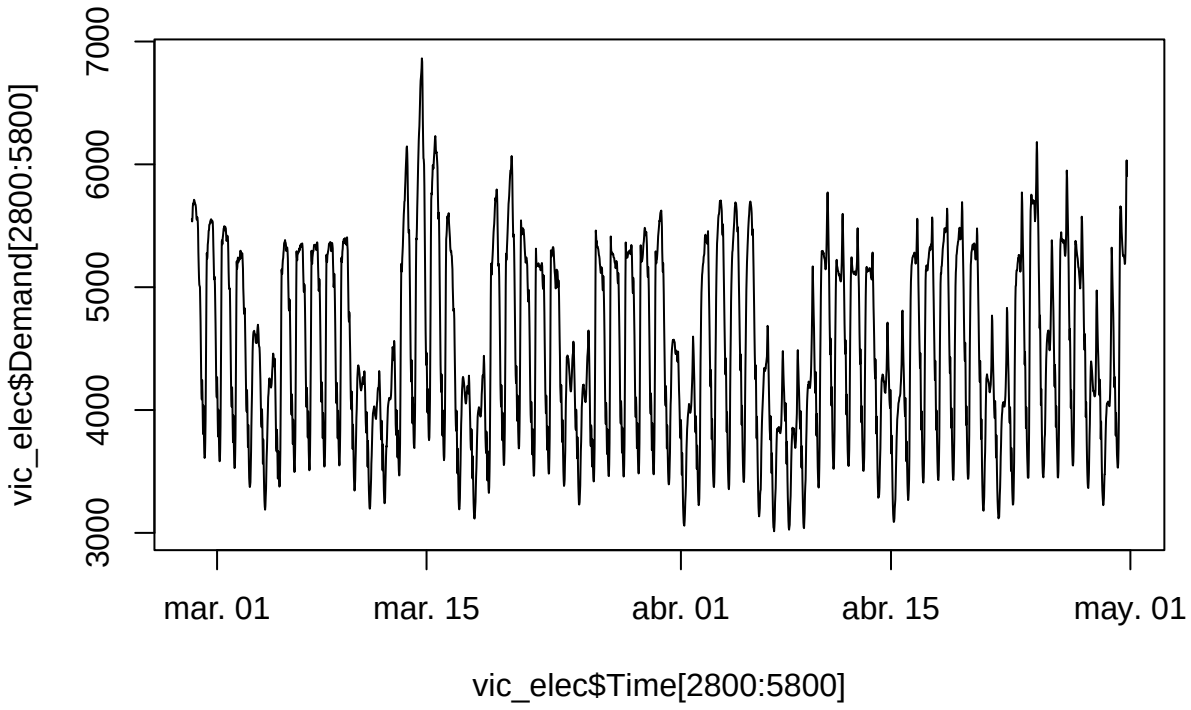
Representemos la serie de demanda eléctrica:

```
plot(vic_elec$Time, vic_elec$Demand, type = "l")
```



En el gráfico anterior se obseva periodicidad anual. Pero también la hay diaria y semanal, aunque es más complicado de advertir en ese gráfico. Representaremos un pequeño tramo (aproximadamente dos meses) de la serie de demanda para ver su comportamiento:

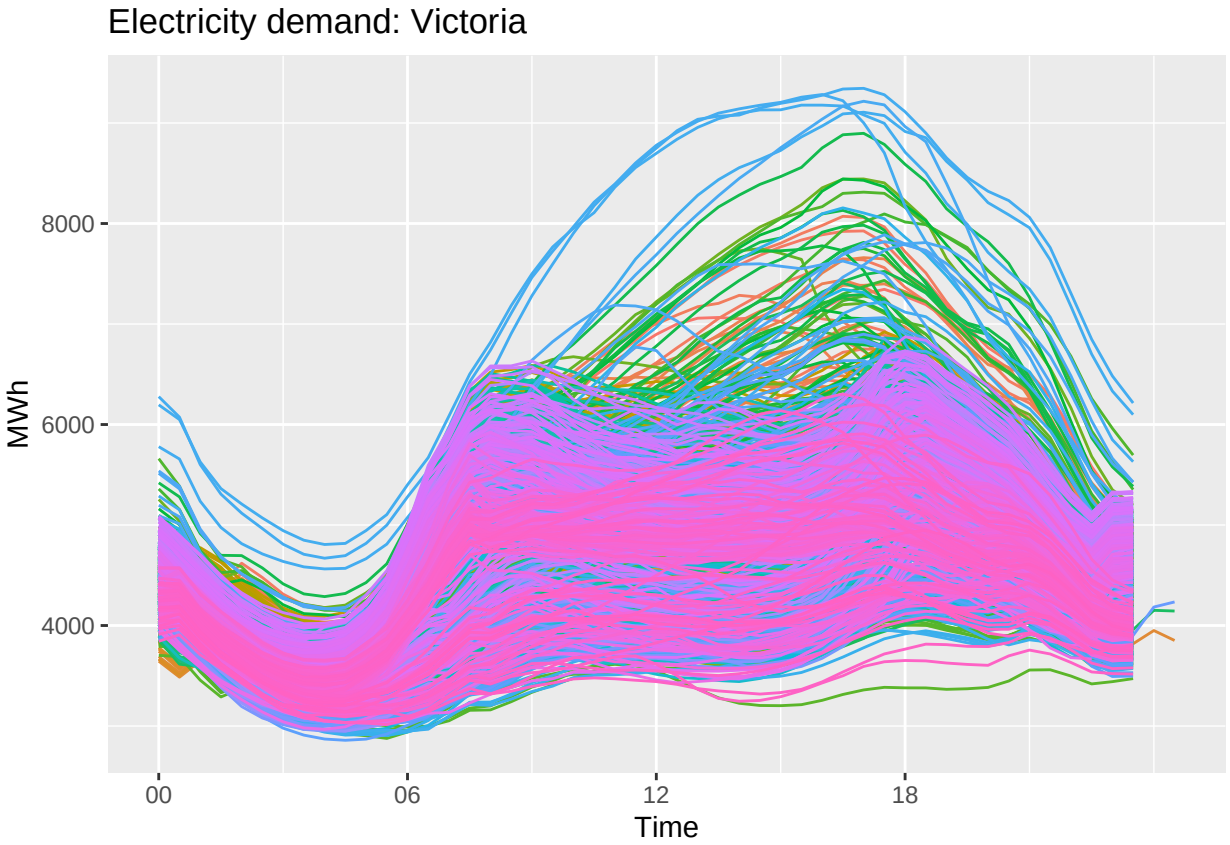
```
plot(vic_elec$Time[2800:5800], vic_elec$Demand[2800:5800], type = "l")
```



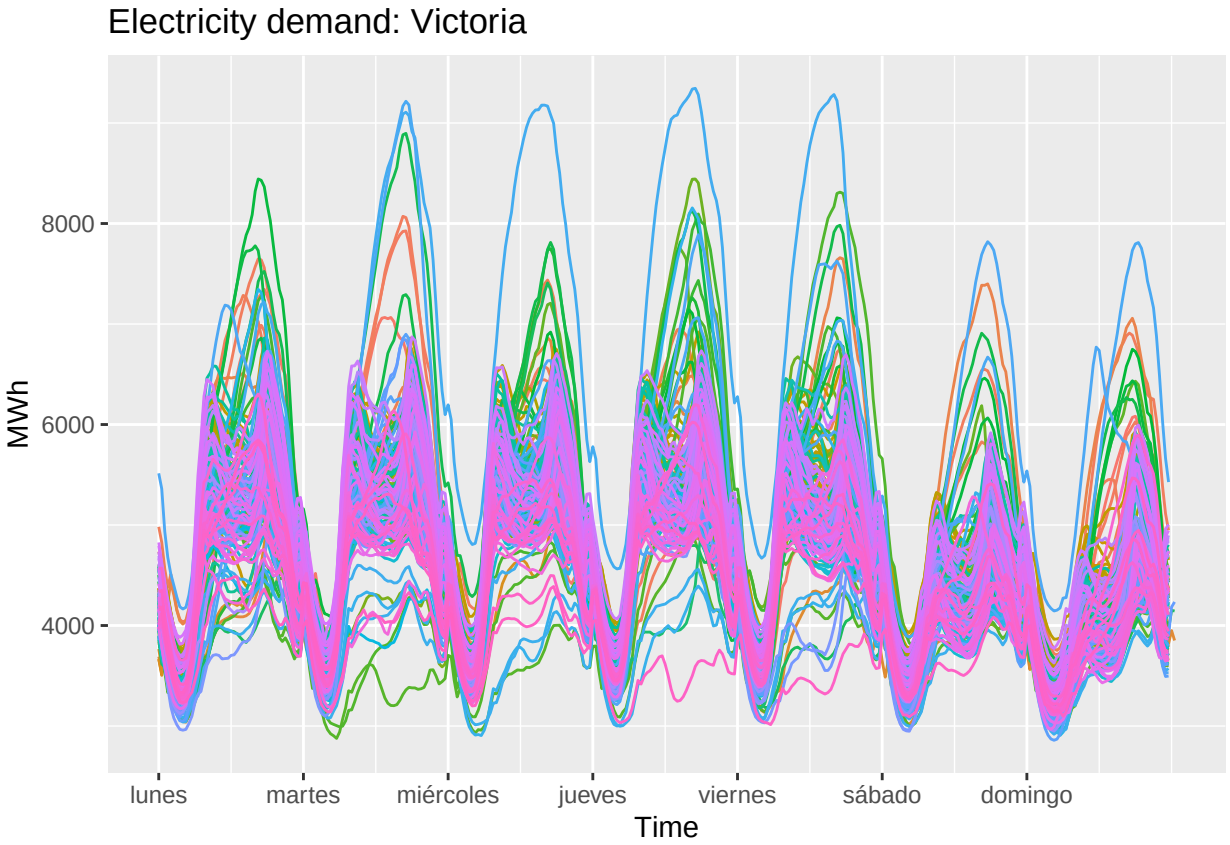
Se observa una clara componente estacional diaria y semanal, además de la anual que ya se mencionó.

Podemos realizar gráficos estacionales para cada periodicidad (diaria, semanal y anual). De esa forma podremos ver patrones en la componente estacional e identificar periodos donde el patrón cambia.

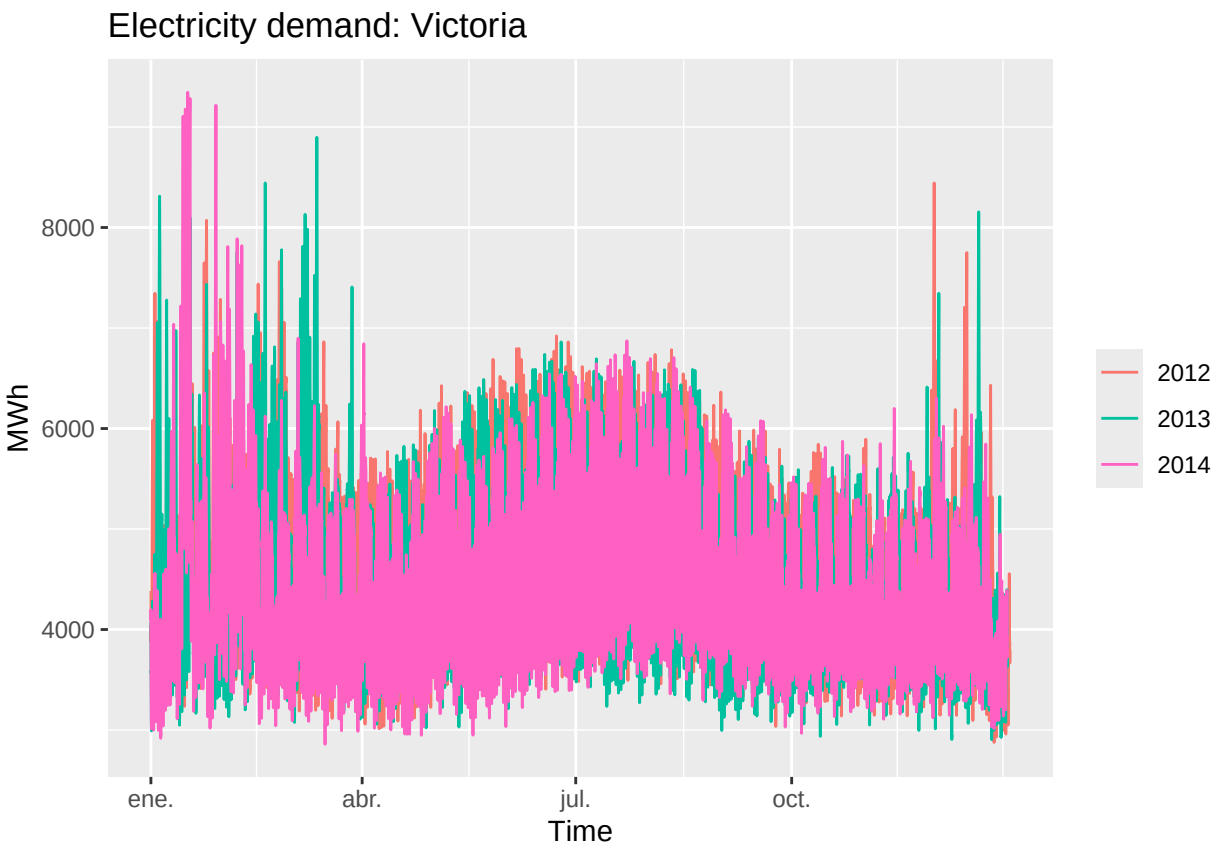
```
vic_elec |> gg_season(Demand, period = "day") +
  theme(legend.position = "none") +
  labs(y="MWh", title="Electricity demand: Victoria")
```




```
vic_elec |> gg_season(Demand, period = "week") +
  theme(legend.position = "none") +
  labs(y="MWh", title="Electricity demand: Victoria")
```




```
vic_elec |> gg_season(Demand, period = "year") +
  labs(y="MWh", title="Electricity demand: Victoria")
```

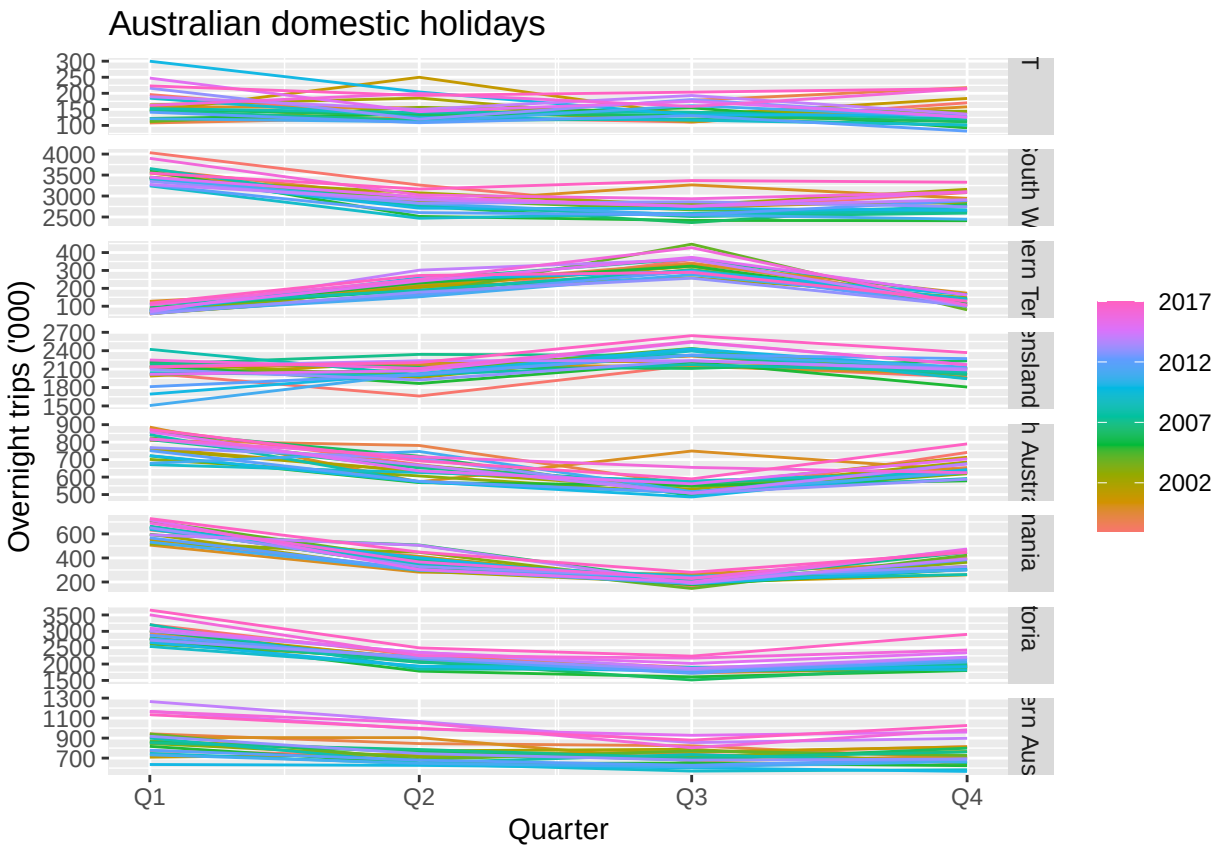


Usemos ahora los datos de número total de turistas por trimestres en diferentes estados de Australia.

```
holidays <- tourism |>
  filter(Purpose == "Holiday") |>
  group_by(State) |>
  summarise(Trips = sum(Trips))
```

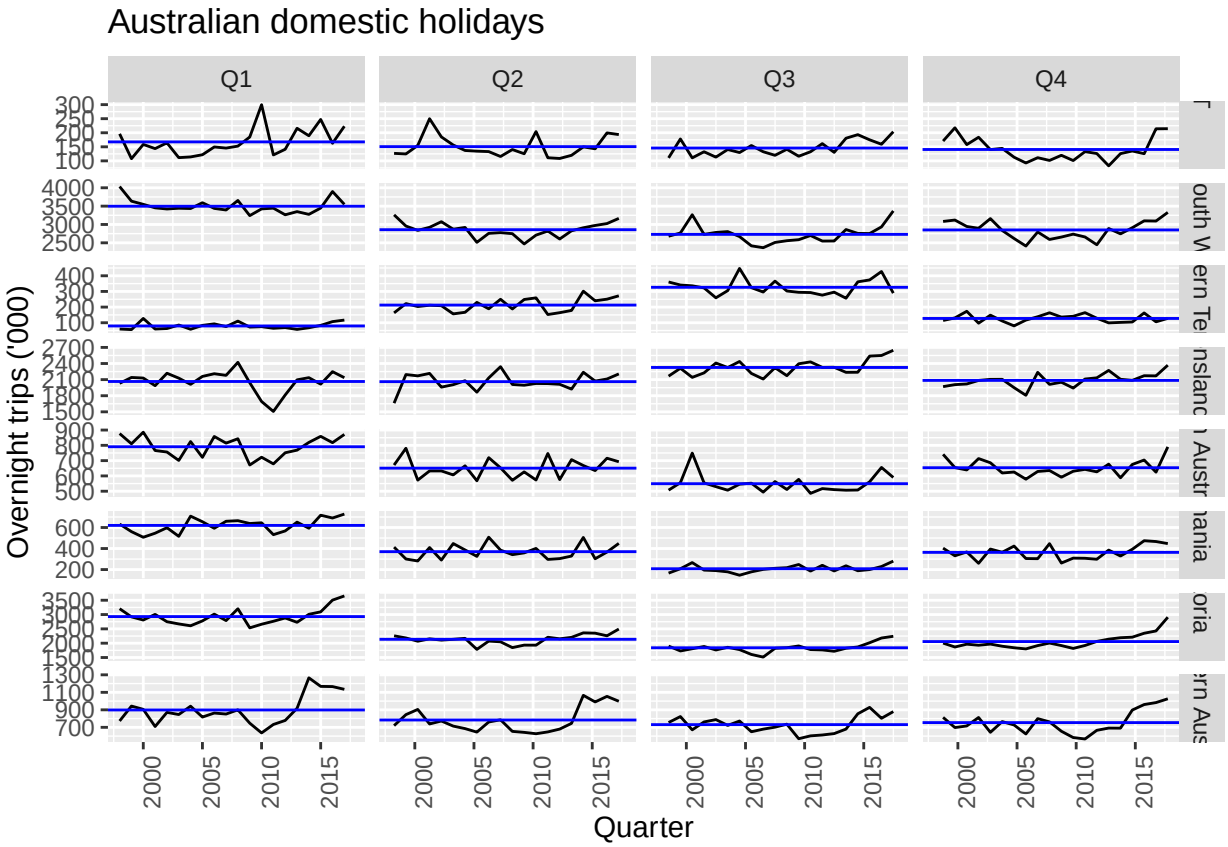
En este caso sólo tenemos estacionalidad de periodo 4 (datos trimestrales). Los gráficos estacionales por estados muestran que los estados del sur (Tasmania, Victoria y South Australia) tienen más turismo en Q1 (su verano), mientras que los estados del norte (Queensland y the Northern Territory) tienen más turismo en Q3 (su estación seca).

```
holidays |> gg_season(Trips) +
  labs(y = "Overnight trips ('000)",
       title = "Australian domestic holidays")
```



También se pueden realizar gráficos estacionales de subseries, donde advertimos que el turismo en Western Australian ha crecido en los últimos años, mientras que el turismo en Victoria ha crecido en los trimestres Q1 y Q4 pero no a mitad de año.

```
holidays |>
  gg_subseries(Trips) +
  labs(y = "Overnight trips ('000)",
       title = "Australian domestic holidays")
```


Australian domestic holidays
Q1
Q2
Q3
Q4
Overnight trips ('000)
Quarter
2000
2005
2010
2015