# Series Temporales: Trabajo Final

*Juanjo Sierra*

*22 de abril de 2019*

## Paquetes a cargar

Importamos los paquetes que necesitamos para resolver los problemas planteados para la práctica.

```
library(tseries)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

## Problema 1

Teniendo los datos de una estación meteorológica (ubicada en Loja) desde mayo de 2013 hasta febrero de 2018, se pide predecir los valores de temperatura máxima mensuales para los meses de marzo y abril de 2018.

En primer lugar, se leen los datos de la estación meteorológica escogida.

```
datosEstacion = read.csv("5582A.csv", sep = ";")
summary(datosEstacion)
```

```
##      Id              Fecha           Tmax           HTmax
##  5582A:1726   2013-05-07:   1   Min.   : 5.80          : 122
##               2013-05-08:   1   1st Qu.:16.38   15:20  :  92
##               2013-05-09:   1   Median :23.80   15:50  :  83
##               2013-05-10:   1   Mean   :24.00   16:00  :  82
##               2013-05-11:   1   3rd Qu.:31.02   15:40  :  78
##               2013-05-12:   1   Max.   :44.30   15:10  :  75
##               (Other)   :1720   NA's   :122     (Other):1194
##       Tmin           HTmin           Tmed           Racha
##  Min.   :-3.70          : 122   Min.   : 2.10   Min.   :14
##  1st Qu.: 5.50   07:20  : 121   1st Qu.:11.00   1st Qu.:27
##  Median :11.40   07:10  : 109   Median :17.60   Median :31
##  Mean   :11.11   07:40  :  99   Mean   :17.56   Mean   :32
##  3rd Qu.:16.30   07:30  :  97   3rd Qu.:23.43   3rd Qu.:36
##  Max.   :27.30   08:00  :  89   Max.   :35.00   Max.   :73
##  NA's   :122     (Other):1089   NA's   :122     NA's   :131
##      HRacha          Vmax           HVmax          TPrec
##          : 131   Min.   : 4.00          : 131   Min.   : 0.000
##  17:50   :  47   1st Qu.:11.00   17:00  :  36   1st Qu.: 0.000
##  17:10   :  32   Median :14.00   17:40  :  33   Median : 0.000
##  15:40   :  31   Mean   :13.82   15:40  :  31   Mean   : 1.069
```

```
##  16:50  :  31   3rd Qu.:16.00   16:50  :  29   3rd Qu.: 0.100
##  18:10  :  31   Max.   :32.00   16:20  :  26   Max.   :63.600
##  (Other):1423   NA's   :131     (Other):1440   NA's   :132
##      Prec1             Prec2             Prec3             Prec4
##  Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000
##  Median : 0.0000   Median : 0.0000   Median : 0.0000   Median : 0.0000
##  Mean   : 0.2668   Mean   : 0.2618   Mean   : 0.2543   Mean   : 0.2758
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000
##  Max.   :31.3000   Max.   :26.7000   Max.   :21.2000   Max.   :32.5000
##  NA's   :88        NA's   :100       NA's   :77        NA's   :83
```

Como solamente vamos a trabajar con la temperatura máxima y necesitamos agruparlas por fecha, nos quedamos únicamente con esas dos columnas.

```
# Nos quedamos con los valores de fecha y temperatura máxima
datosEstacion = datosEstacion[,2:3]
head(datosEstacion)
```

```
##        Fecha Tmax
## 1 2013-05-07 27.8
## 2 2013-05-08 29.3
## 3 2013-05-09 27.1
## 4 2013-05-10 26.6
## 5 2013-05-11 26.8
## 6 2013-05-12 26.1
```

Como hemos comprobado antes en el summary que sí que hay valores perdidos (122), vamos a eliminar las instancias en las que haya un NA para poder calcular correctamente cuál es la temperatura máxima en cada mes.

```
# Elimino aquellos datos que sean NA
datosEstacionSinNA = datosEstacion[-which(is.na(datosEstacion$Tmax)),]

# Comprobamos la nueva dimensionalidad de los datos
# y confirmamos que ya no hay datos perdidos
dim(datosEstacionSinNA)
```

```
## [1] 1604    2
```

```
anyNA(datosEstacionSinNA)
```

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

Hemos reducido la dimensionalidad de los datos de 1726 a 1604, pero ya no hay ningún valor perdido y podemos agrupar los datos por mes, obteniendo la temperatura máxima en cada uno.

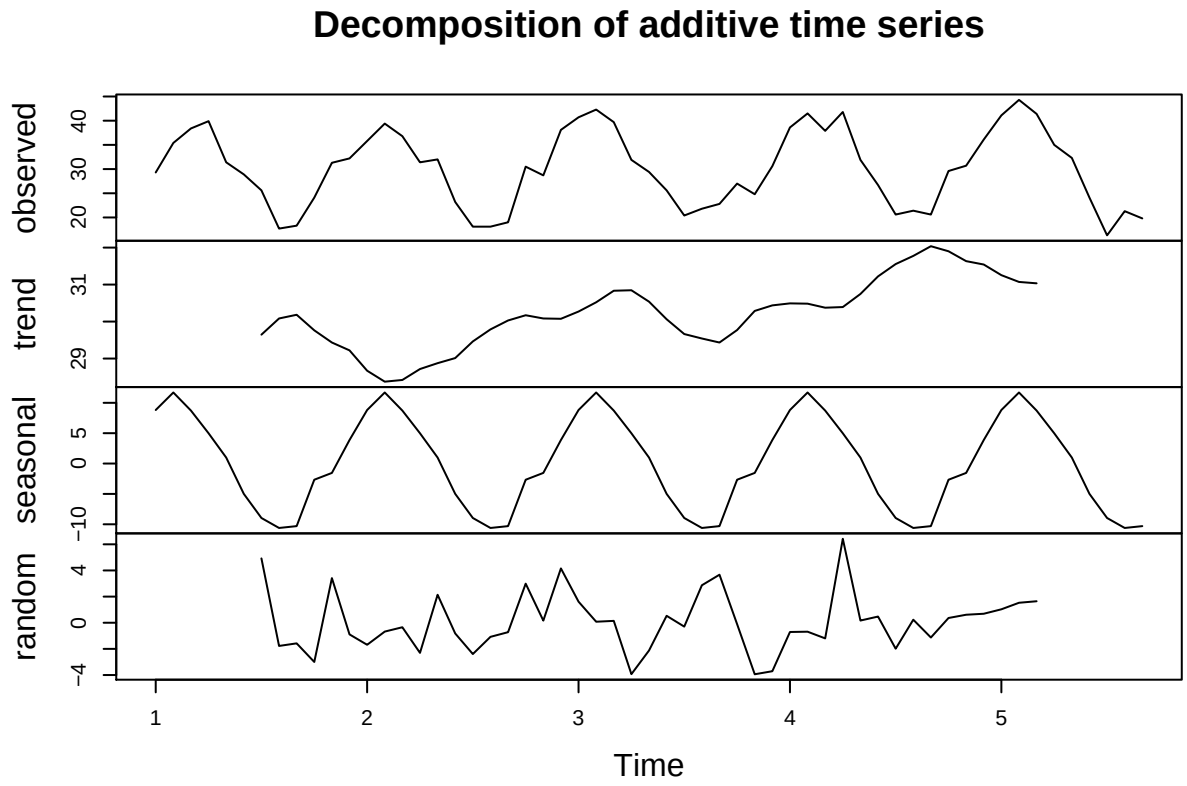
Para obtener dichos datos máximos utilizo la librería `dplyr`.

```
# Almacenamos la variable Fecha como tipo Date
datosEstacionSinNA$Fecha = as.Date(datosEstacionSinNA$Fecha)

# Convertimos el dataset en dos columnas (Fecha y Mes),
# agrupamos por mes y año y para todos los valores nos
# quedamos con el valor máximo de TMax
datosTmaxMensual = datosEstacionSinNA %>%
mutate(Mes = format(Fecha, "%m"), Año = format(Fecha, "%Y")) %>%
group_by(Año, Mes) %>%
summarise(Tmax = max(Tmax))
```

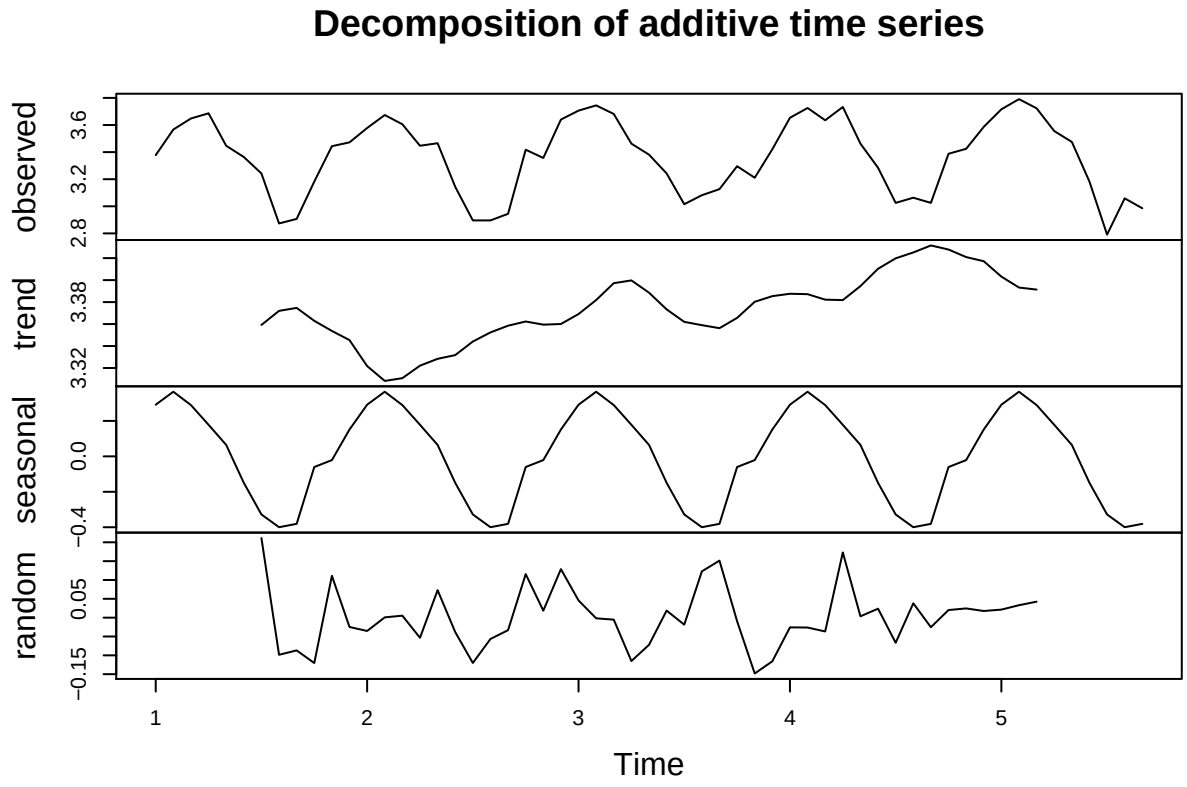
Ahora podemos trabajar con la columna TMax como nuestra serie temporal. Podemos crear el objeto "Serie Temporal" con la librería `tseries`. Usando `plot` y `decompose` se pueden echar un vistazo general al aspecto de nuestros datos. Incluimos un valor de `frequency` de 12 porque es lo que estimamos que es el periodo de la estacionalidad (de año en año y tenemos valores mensuales).

```
# Observamos la tendencia y estacionalidad con el decompose
# y utilizamos frecuencia 12 porque asumimos que tienen
# estacionalidad anual
serie = datosTmaxMensual$Tmax
serie.ts = ts(serie, frequency=12)
plot(decompose(serie.ts))
```




Buscando que la serie sea estacionaria, implicando eso que no varíe ni en media ni en varianza, vamos a realizar una transformación logarítmica para conseguir suavizar los valores.

```
# Realizamos una transformación logarítmica
# y volvemos a mostrar los valores del decompose
serie.ts.log = log(serie.ts)
serie.log = log(serie)
plot(decompose(serie.ts.log))
```




Ya podemos empezar a trabajar con los datos. Como primer paso, vamos a asumir que todo el conjunto de train es el total de datos que tenemos y que la predicción a realizar estará compuesta por 2 valores (marzo y abril de 2018).

```
nPred = 2
serie.train = serie.log
tiempo.train = 1:length(serie.train)
tiempo.pred = (length(tiempo.train)+1):(length(tiempo.train)+nPred)
```

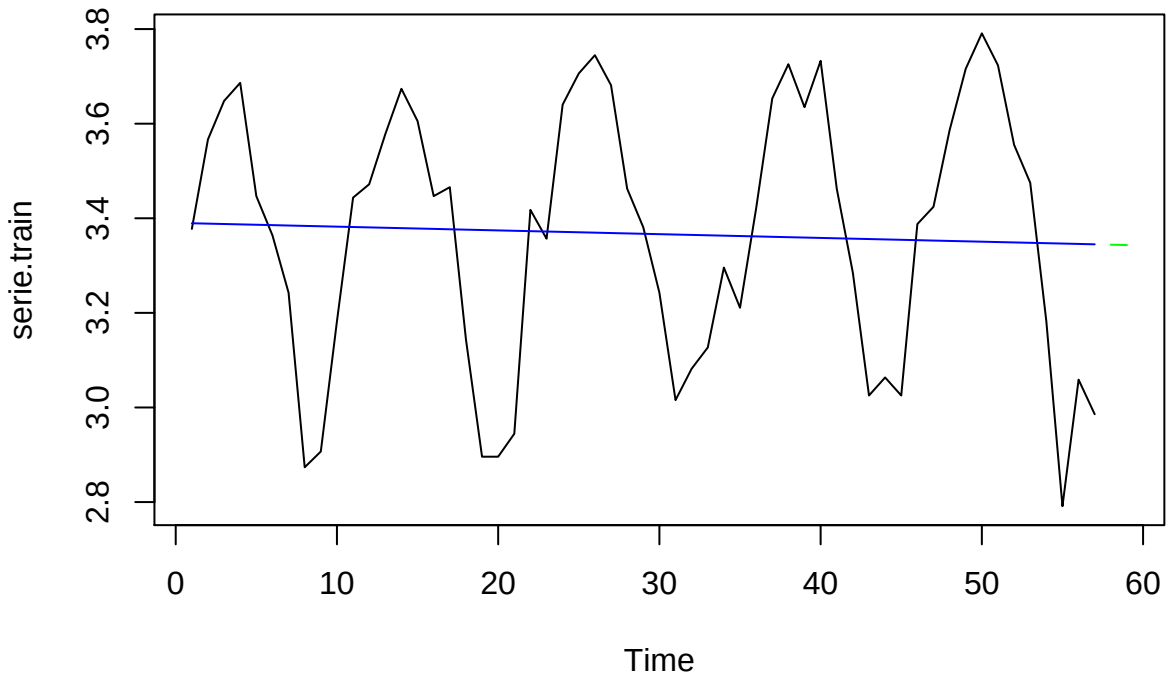
A continuación tenemos que estimar la tendencia. Vamos a utilizar un modelo lineal simple dado que basándonos en la gráfica puede generalizar aceptablemente. Construimos el modelo lineal con la función `lm`.

```
parametros.lm = lm(serie.train ~ tiempo.train)

tendencia.train = parametros.lm$coefficients[1]+tiempo.train*parametros.lm$coefficients[2]
tendencia.pred = parametros.lm$coefficients[1]+tiempo.pred*parametros.lm$coefficients[2]
```

En la siguiente gráfica mostramos la tendencia estimada en la misma gráfica que la serie temporal.

```
plot.ts(serie.train, xlim=c(1, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.train, tendencia.train, col="blue")
lines(tiempo.pred, tendencia.pred, col="green")
```




Para validar que el modelo es correcto, dado que no se puede afirmar con la gráfica que hemos obtenido, utilizaremos el test de Jarque Bera sobre los residuos que han quedado de generar el modelo lineal.

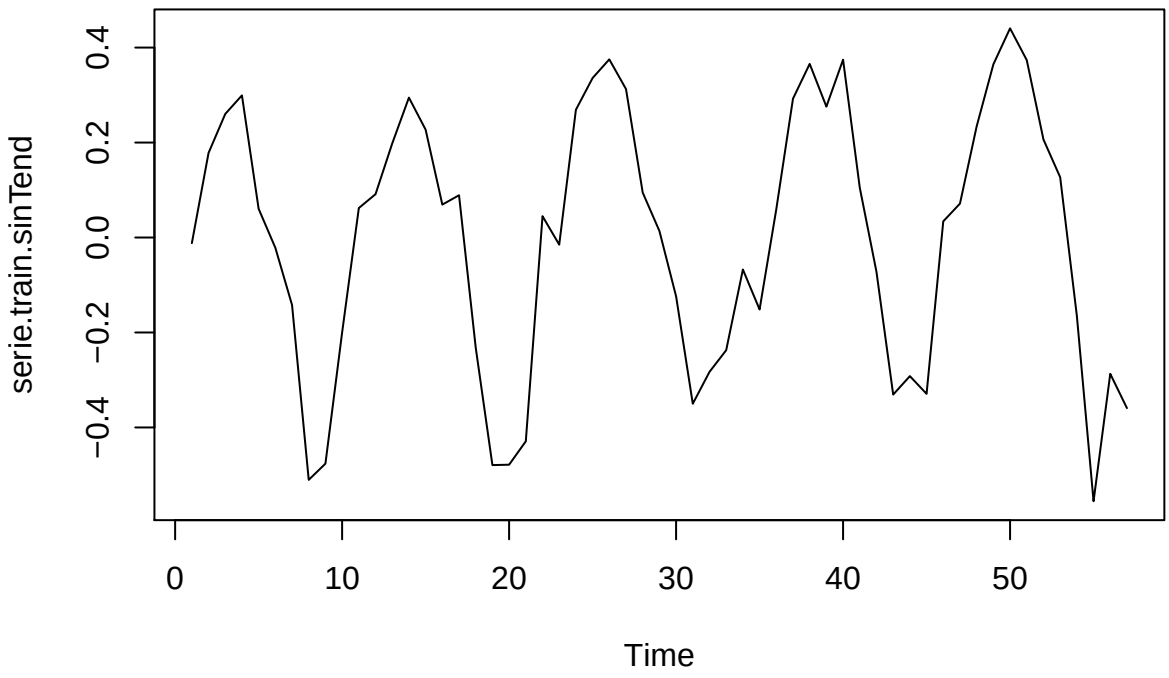
```
JB.train = jarque.bera.test(parametros.lm$residuals)
JB.train
```

```
##
##  Jarque Bera Test
##
## data:  parametros.lm$residuals
## X-squared = 3.3983, df = 2, p-value = 0.1828
```

El test de Jarque Bera da un p-value de 0.18, que está por encima de 0.05. Por este motivo, no tenemos suficiente confianza como para afirmar que los residuos no sigan una distribución normal, y por tanto asumimos que sí que la siguen.

El siguiente paso es eliminar la tendencia a la serie. Comprobaremos en una gráfica qué aspecto tiene una vez eliminada esa tendencia.

```
serie.train.sinTend = serie.train - tendencia.train
plot.ts(serie.train.sinTend, xlim=c(1, tiempo.train[length(tiempo.train)]))
```




Se puede observar que la gráfica tiene el mismo aspecto, solamente se ha trasladado en el eje Y, ubicándose ahora en torno al 0.

El siguiente paso es eliminar la estacionalidad del modelo. Inicialmente supusimos una estacionalidad de frecuencia 12 (anual).

```
k = 12
estacionalidad = decompose(serie.ts.log)$seasonal[1:k]
estacionalidad
```

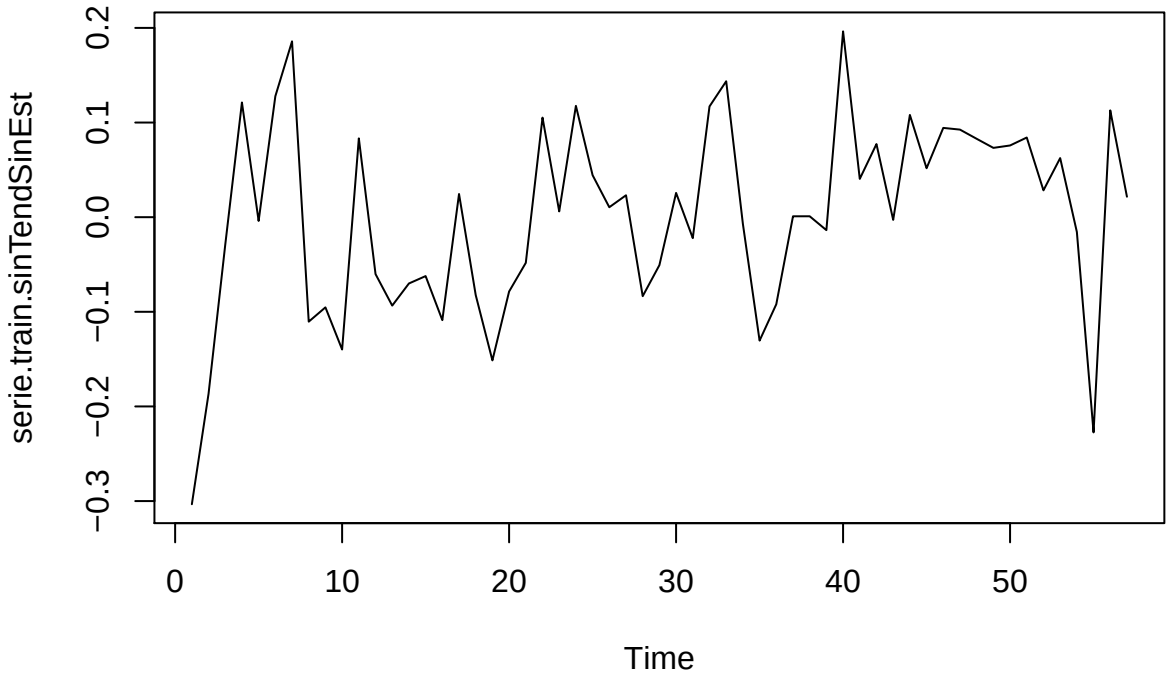
```
##  [1]  0.29147951  0.36470197  0.28941527  0.17808018  0.06450275
##  [6] -0.14945909 -0.32790964 -0.39990923 -0.38087543 -0.06025461
## [11] -0.02116616  0.15139447
```

Aquí vemos los 12 valores que componen la estacionalidad de la serie. Para eliminar la estacionalidad hay que restar estos valores de forma periódica a lo largo de la serie.

```
# Aprovecho el reciclaje de R para no tener que crear una variable auxiliar
serie.train.sinTendSinEst = serie.train.sinTend - estacionalidad
```

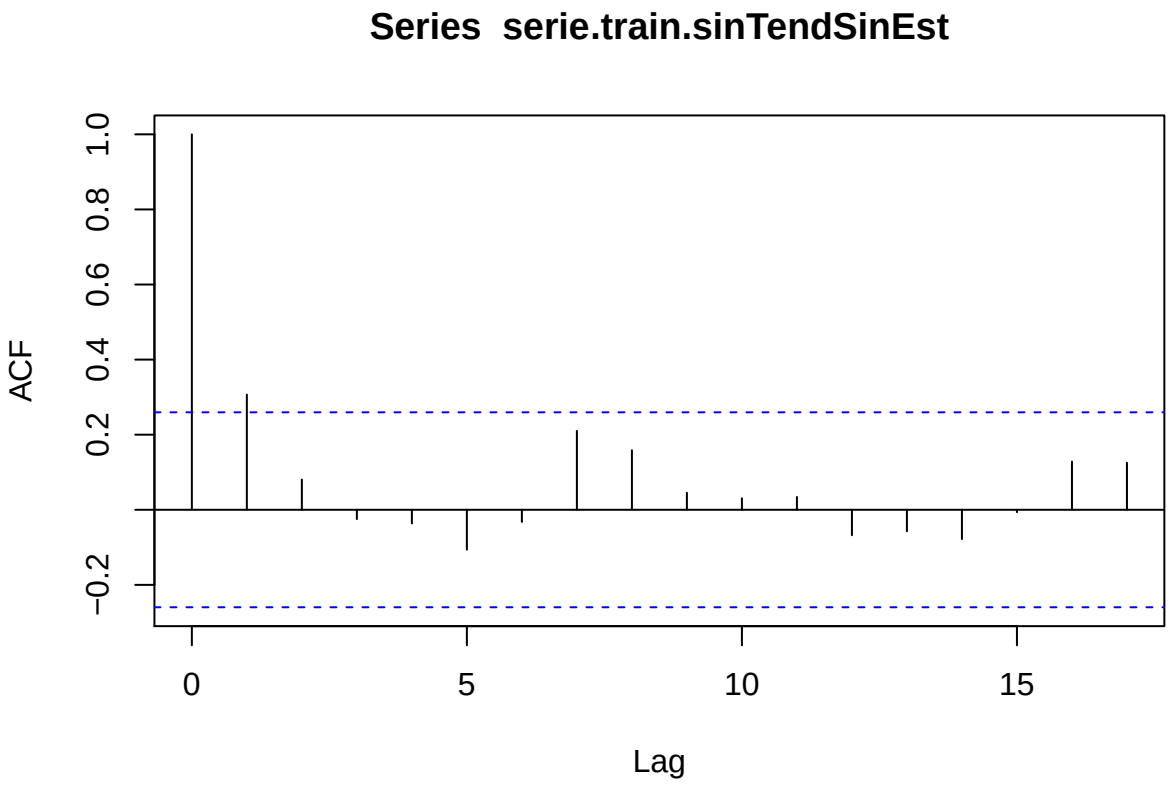
```
## Warning in serie.train.sinTend - estacionalidad: longitud de objeto mayor
## no es múltiplo de la longitud de uno menor
```

```
plot.ts(serie.train.sinTendSinEst, xlim=c(1, tiempo.train[length(tiempo.train)]))
```




Esta serie ya no tiene tendencia ni estacionalidad, como íbamos buscando. Ahora queremos comprobar si es estacionaria. Para ello podemos comenzar por realizar un test ACF.

```
acf(serie.train.sinTendSinEst)
```



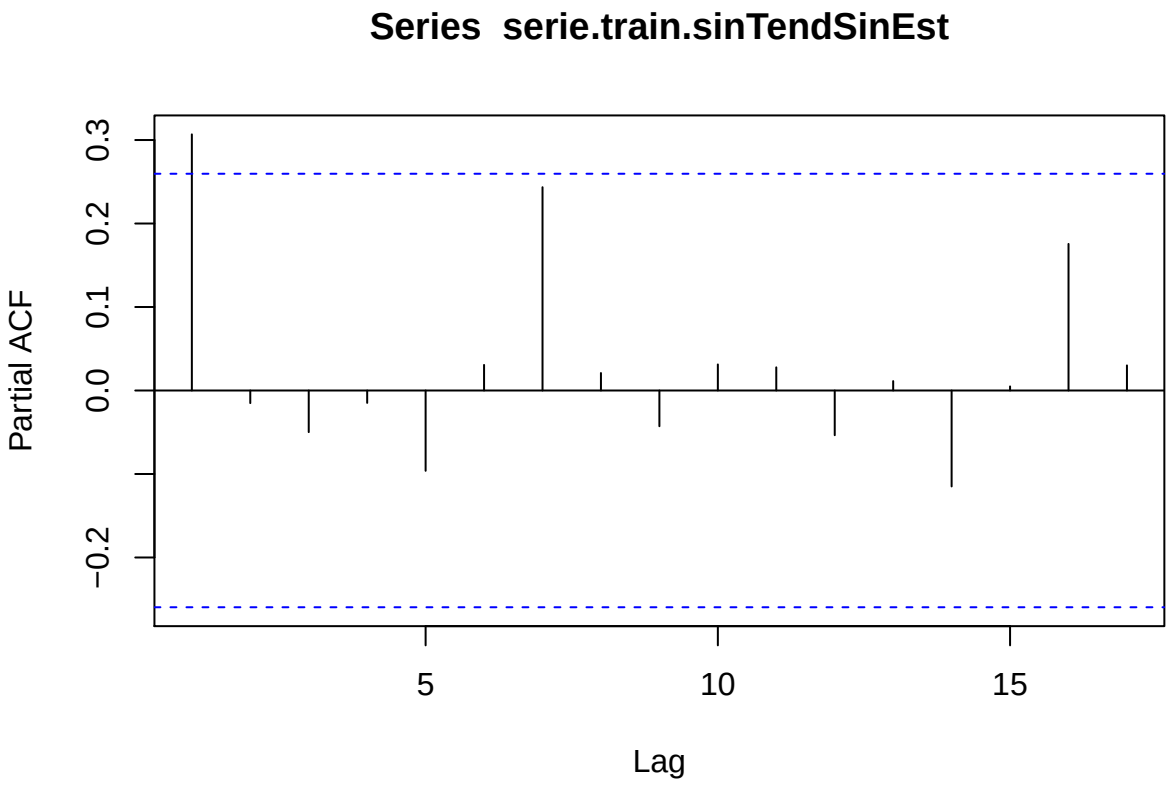

En la gráfica podemos observar que los valores descienden muy rápido y continúan por la serie siguiendo un formato de "olas", por lo que tiene toda la pinta de ser un modelo autorregresivo. Para asegurar que se trata de una serie estacionaria vamos a realizar el test de Dickey-Fuller.

```
adf.serie.train = adf.test(serie.train.sinTendSinEst)
adf.serie.train
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  serie.train.sinTendSinEst
## Dickey-Fuller = -3.8728, Lag order = 3, p-value = 0.02147
## alternative hypothesis: stationary
```

Según los resultados del test de Dickey-Fuller, tenemos una confianza de en torno al 98% de que la serie sea estacionaria. Debido a que hemos observado este comportamiento de "olas" en el test ACF podemos sugerir que se trata de un modelo autorregresivo, pero podemos comprobar en un test PACF si también puede tratarse de un modelo de medias móviles.

```
pacf(serie.train.sinTendSinEst)
```




En esta gráfica se intuye un comportamiento similar pero sin embargo los valores más lejanos al 0 se encuentran muy próximos al umbral y no se produce un descenso tan rápido y tan grande en estos valores. Por lo tanto, propongo generar un modelo autorregresivo de grado 0 (el último instante de tiempo en el que el PACF pasa del umbral) y sin diferenciación.

```
modelo = arima(serie.train.sinTendSinEst, order = c(0,0,0))
```

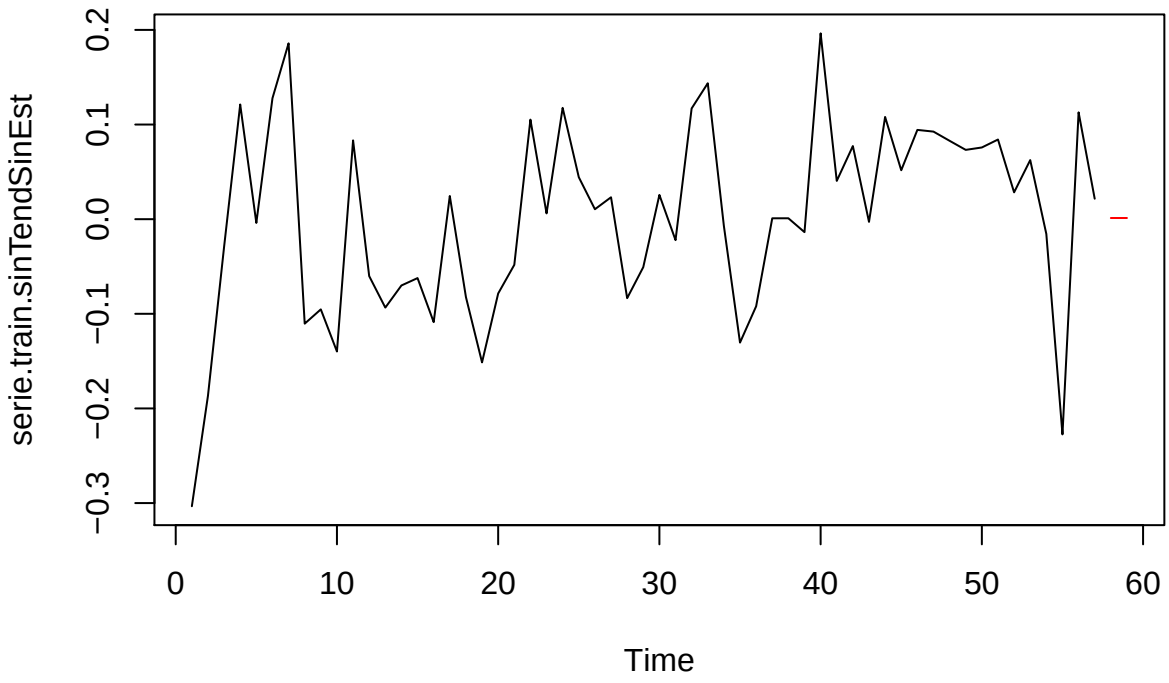
Una vez construido el modelo podemos generar las predicciones para los valores que buscamos (los 2 siguientes, marzo y abril).

```
predicciones = predict(modelo, n.ahead = nPred)
valoresPredichos = predicciones$pred
valoresPredichos
```

```
## Time Series:
## Start = 58
## End = 59
## Frequency = 1
## [1] 0.001227609 0.001227609
```

Podemos observar los valores predichos con el resto de los valores de la serie transformada.

```
plot.ts(serie.train.sinTendSinEst, xlim=c(1, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.pred, valoresPredichos, col="red")
```




Se aprecia que al menos los valores siguen la tendencia que parecía intuirse de los últimos valores (hay un pico que está decayendo en los últimos meses).

Para comprobar que este modelo sea válido vamos a aplicar el test de Box-Pierce para comprobar que los residuos que quedan del modelo siguen una distribución aleatoria.

```
boxPierce.test = Box.test(modelo$residuals)
boxPierce.test
```

```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 5.3652, df = 1, p-value = 0.02054
```

Según el p-value que refleja el test de Box-Pierce los residuos no siguen una distribución aleatoria, es decir, hay al menos una parte de la serie que se puede modelar mediante algo que no sea aleatorio. Por este motivo vamos a volver a las gráficas ACF y PACF y vamos a comprobar que, de considerar un modelo de medias móviles en lugar de uno autorregresivo, podríamos generar un modelo ARIMA(0,0,1). Probemos si con este modelo las predicciones y los residuos que nos quedan se adaptan a lo que buscamos.

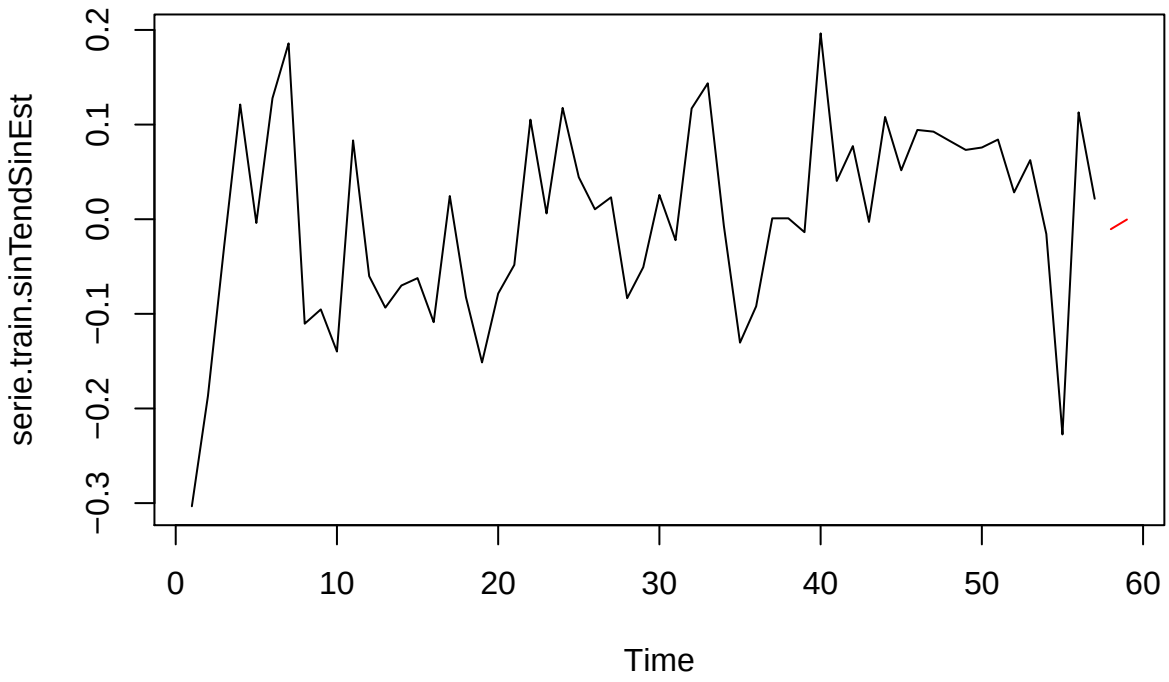
```
modelo = arima(serie.train.sinTendSinEst, order = c(0,0,1))

predicciones = predict(modelo, n.ahead = nPred)
valoresPredichos = predicciones$pred
valoresPredichos
```

```
## Time Series:
## Start = 58
```

```
## End = 59
## Frequency = 1
## [1] -0.0105291868 -0.0003097663
```

```
plot.ts(serie.train.sinTendSinEst, xlim=c(1, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.pred, valoresPredichos, col="red")
```




A primera vista esta aproximación también podría encajar. Vamos a probar el test de Box-Pierce.

```
boxPierce.test = Box.test(modelo$residuals)
boxPierce.test
```

```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 0.00025506, df = 1, p-value = 0.9873
```

Este test ya sí nos asegura que los residuos son aleatorios, por lo que el modelo se adapta mejor a la serie. Con los tests de Jarque Bera y Shapiro-Wilk probamos si los residuos siguen una distribución normal.

```
jarqueBera.test = jarque.bera.test(modelo$residuals)
jarqueBera.test
```

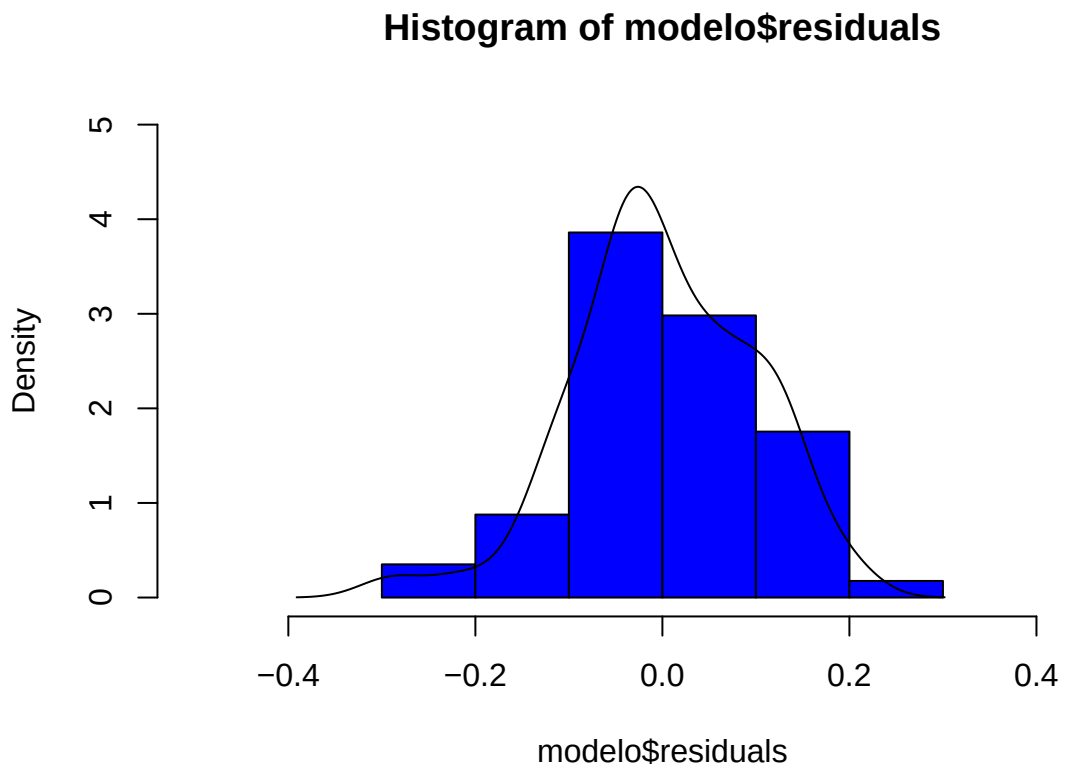
```
##
##  Jarque Bera Test
##
## data:  modelo$residuals
## X-squared = 1.2443, df = 2, p-value = 0.5368
```

```
shapiroWilk.test = shapiro.test(modelo$residuals)
shapiroWilk.test
```

```
##
##  Shapiro-Wilk normality test
##
## data:  modelo$residuals
## W = 0.97897, p-value = 0.4209
```

Los p-values tan altos que dan estos tests no dan lugar a pensar que los residuos puedan seguir una distribución no normal. Por tanto, estos residuos también se ajustan a lo que buscamos y podemos asumir que el modelo es bueno. Podemos mostrar un histograma con la distribución de los residuos para verlo gráficamente.

```
hist(modelo$residuals, col="blue", prob=TRUE, ylim=c(0,5), xlim=c(-0.5,0.5))
lines(density(modelo$residuals))
```




Para obtener los datos finales de temperatura hay que deshacer sobre los datos predichos las transformaciones que hemos realizado antes.

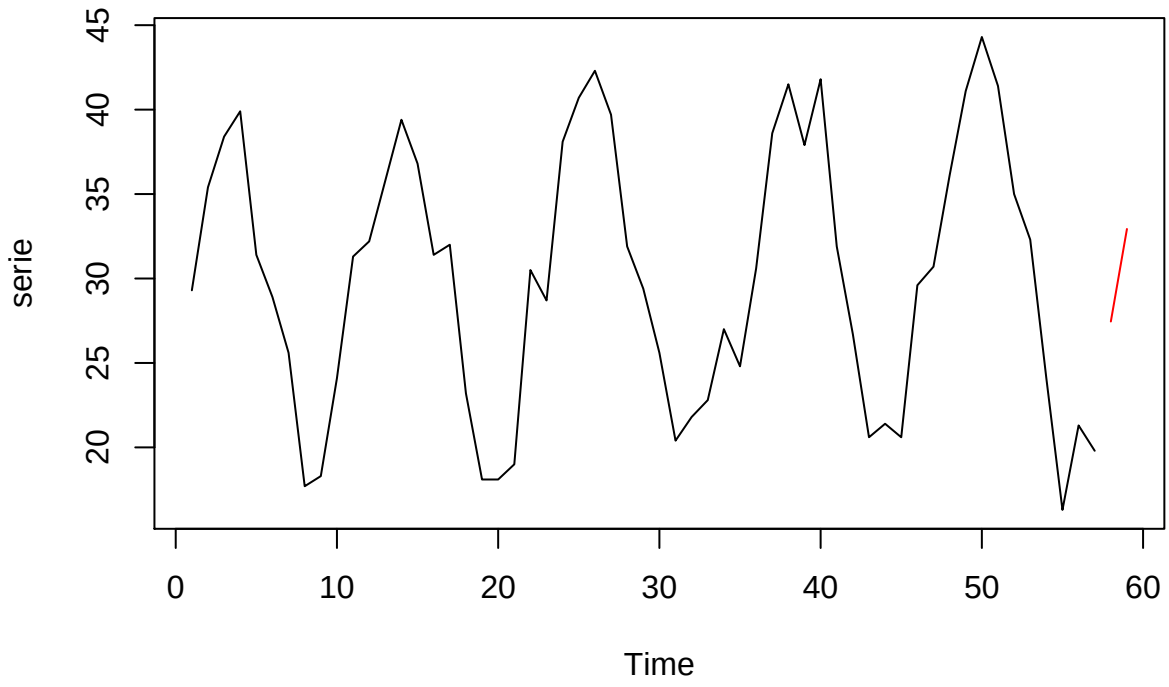
```
# Sumamos estacionalidad
# De nuevo aprovecho el reciclaje de R
# Como el vector de estacionalidad empieza en mayo
# hay que especificar que el valor que sumamos es el que
# le corresponde a marzo y abril
valoresPredichos.Est = valoresPredichos + estacionalidad[11:12]

# Sumamos tendencia
valoresPredichos.EstTend = valoresPredichos.Est + tendencia.pred
```

```
# Deshacemos transformación logarítmica
valoresPredichos.EstTendExp = exp(valoresPredichos.EstTend)
```

Por último, mostramos los datos predichos junto a los que hemos utilizado para entrenar, de forma que continúen la serie.

```
plot.ts(serie, xlim=c(1, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.pred, valoresPredichos.EstTendExp, col="red")
```




Los valores finales predichos por el modelo para los meses de marzo y abril son los siguientes.

```
valoresPredichos.EstTendExp
```

```
## Time Series:
## Start = 58
## End = 59
## Frequency = 1
## [1] 27.45242 32.93183
```

## Problema 2

Teniendo los datos de una estación meteorológica (ubicada en Loja) desde mayo de 2013 hasta febrero de 2018, de forma diaria, se pide predecir los valores de temperatura máxima por día en la primera semana de marzo de 2018.

Partiendo de los datos que tenemos de la estación, solamente sabiendo la fecha y la temperatura máxima, imputamos los valores perdidos para poder tener una serie continua.

```r
library(imputeTS)
```

```
##
## Attaching package: 'imputeTS'
```

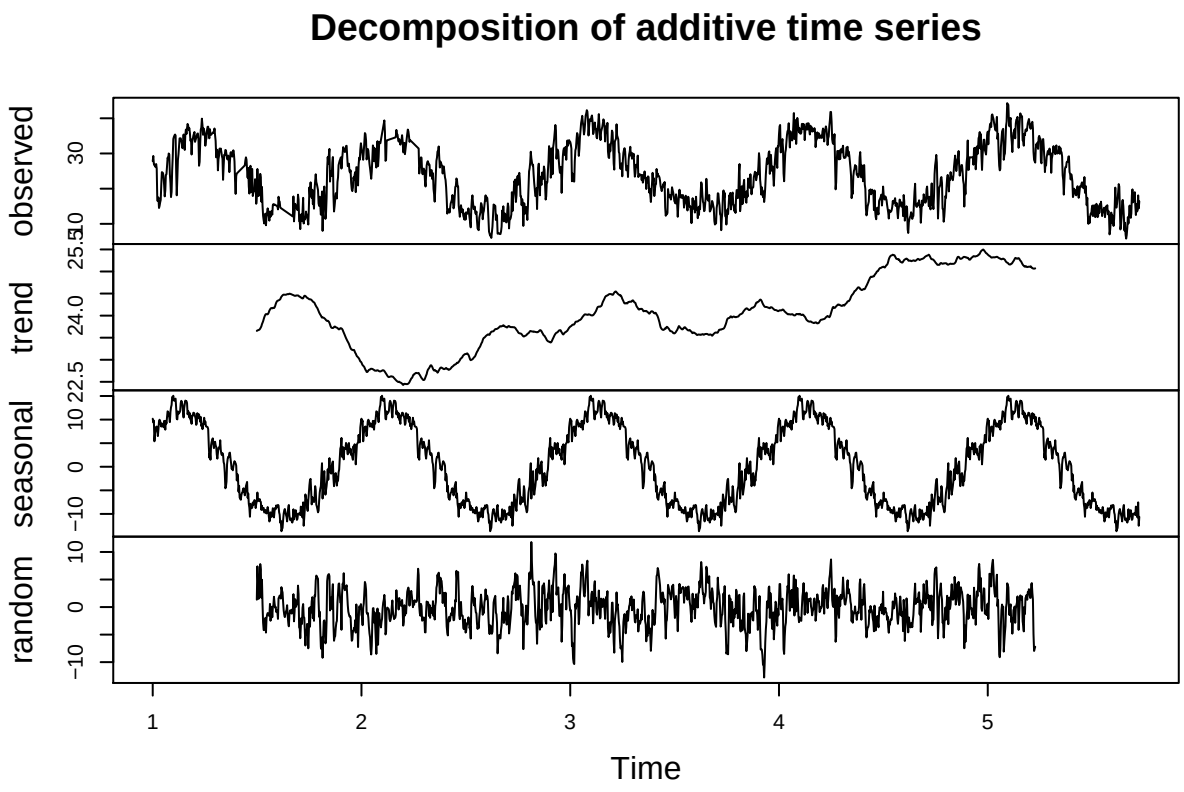
```
## The following object is masked from 'package:tseries':
##
##     na.remove
```

```r
datosEstacionImp = na.interpolation(datosEstacion)
anyNA(datosEstacionImp)
```

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

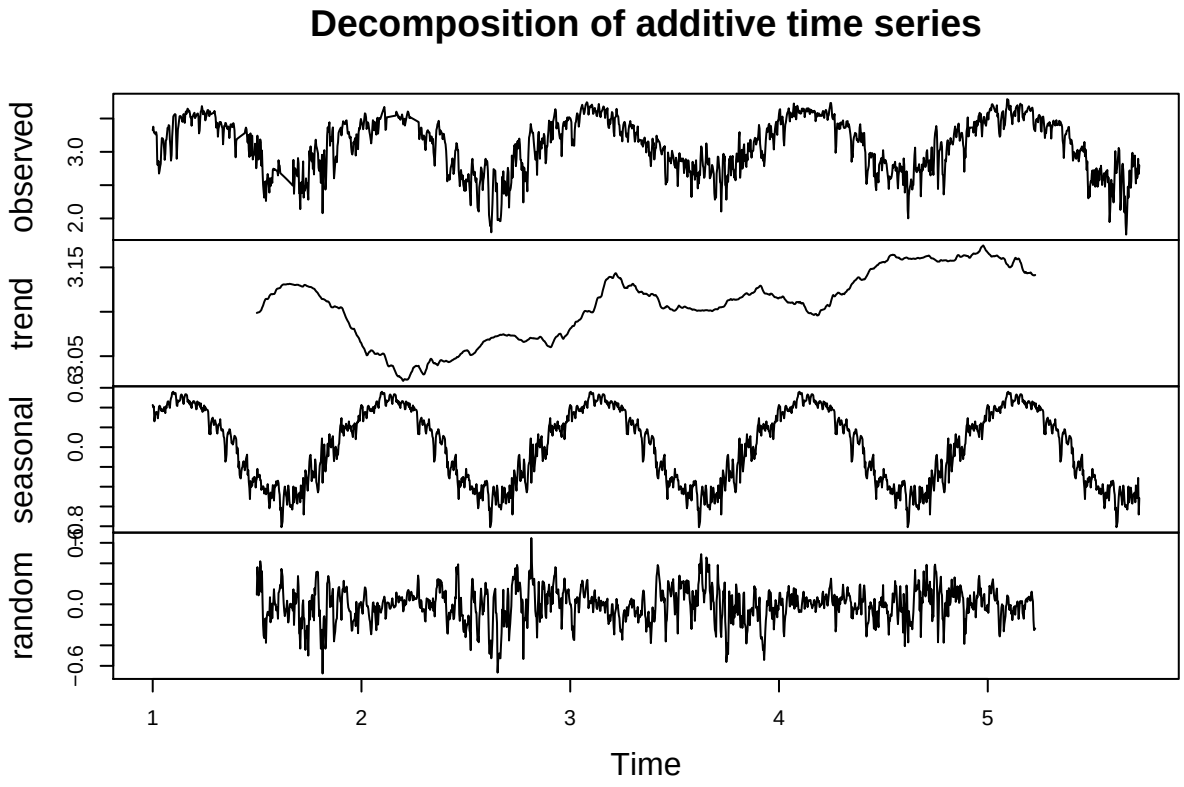
Una vez imputados los datos analizamos la nueva serie de datos diarios con la función `decompose`. Esta vez, al tratar con datos diarios, la frecuencia de la estacionalidad será 365.

```r
serie = datosEstacionImp$Tmax
serie.ts = ts(serie, frequency=365)
plot(decompose(serie.ts))
```




Para suavizar los datos de la componente aleatoria y acercar su media al valor 0, vamos a realizar una transformación logarítmica a la serie.

```r
# Realizamos una transformación logarítmica
# y volvemos a mostrar los valores del decompose
serie.ts.log = log(serie.ts)
serie.log = log(serie)
plot(decompose(serie.ts.log))
```




Ahora los datos se aproximan al 0 y la media saldrá muy próxima a ese valor. Está más próxima a ser estacionaria.

Vamos a calcular los datos a predecir y a definir la serie que usaremos como train (todos los valores). El número de predicciones será 7 (los 7 días de la primera semana de marzo).

```
nPred = 7
serie.train = serie.log
tiempo.train = 1:length(serie.train)
tiempo.pred = (length(tiempo.train)+1):(length(tiempo.train)+nPred)
```

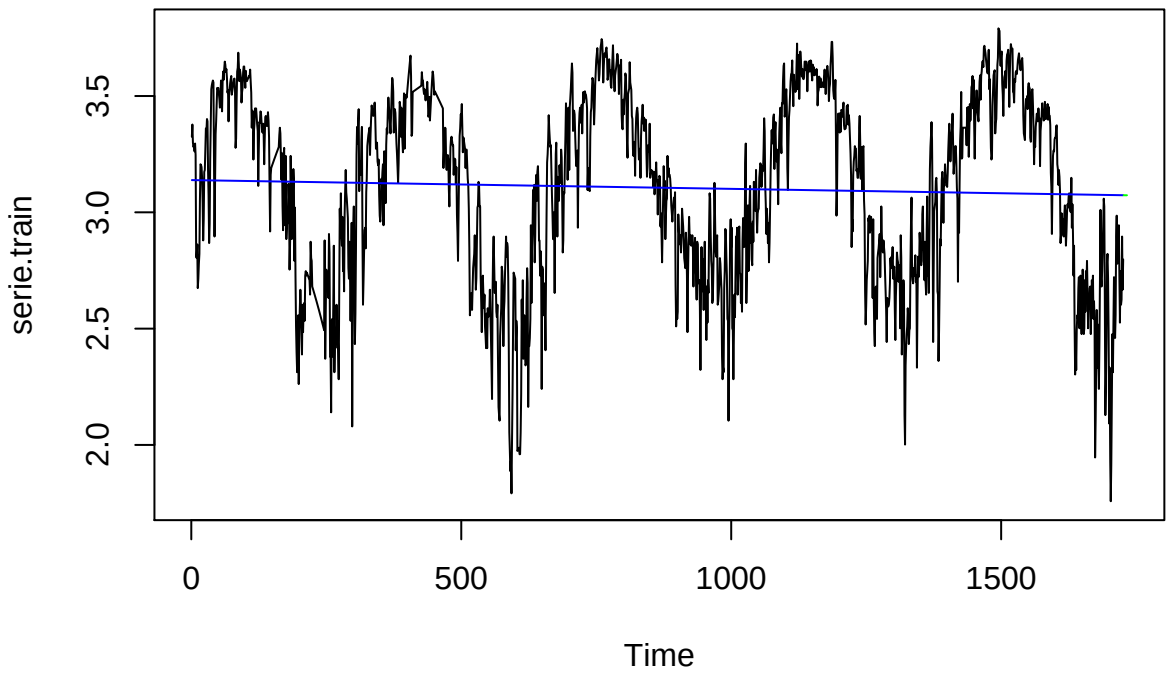
El siguiente paso es estimar la tendencia de la serie.

```
parametros.lm = lm(serie.train ~ tiempo.train)

tendencia.train = parametros.lm$coefficients[1]+tiempo.train*parametros.lm$coefficients[2]
tendencia.pred = parametros.lm$coefficients[1]+tiempo.pred*parametros.lm$coefficients[2]
```

En la siguiente gráfica mostramos la tendencia estimada en la misma gráfica que la serie temporal.

```
plot.ts(serie.train, xlim=c(1, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.train, tendencia.train, col="blue")
lines(tiempo.pred, tendencia.pred, col="green")
```




Vamos a utilizar el test de Jarque Bera sobre los residuos que han quedado de generar el modelo lineal, para comprobar si siguen una distribución normal.

```
JB.train = jarque.bera.test(parametros.lm$residuals)
JB.train
```

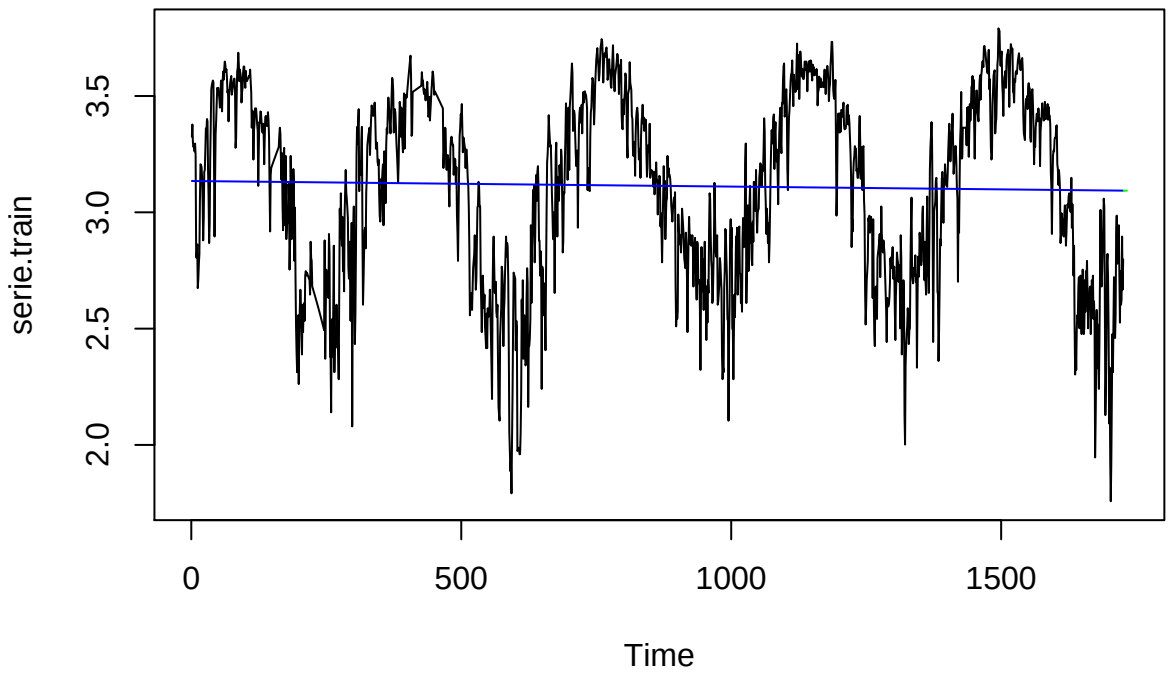
```
##
##  Jarque Bera Test
##
## data:  parametros.lm$residuals
## X-squared = 89.808, df = 2, p-value < 2.2e-16
```

Como el test de Jarque Bera no indica que los residuos sigan una distribución normal, vamos a probar un nuevo modelo que haga transformaciones al atributo tiempo.

```
parametros.lm = lm(serie.train ~ tiempo.train + I(tiempo.train^2))

tendencia.train = parametros.lm$coefficients[1]+tiempo.train*parametros.lm$coefficients[2]
tendencia.pred = parametros.lm$coefficients[1]+tiempo.pred*parametros.lm$coefficients[2]

plot.ts(serie.train, xlim=c(1, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.train, tendencia.train, col="blue")
lines(tiempo.pred, tendencia.pred, col="green")
```




```
JB.train = jarque.bera.test(parametros.lm$residuals)
JB.train
```

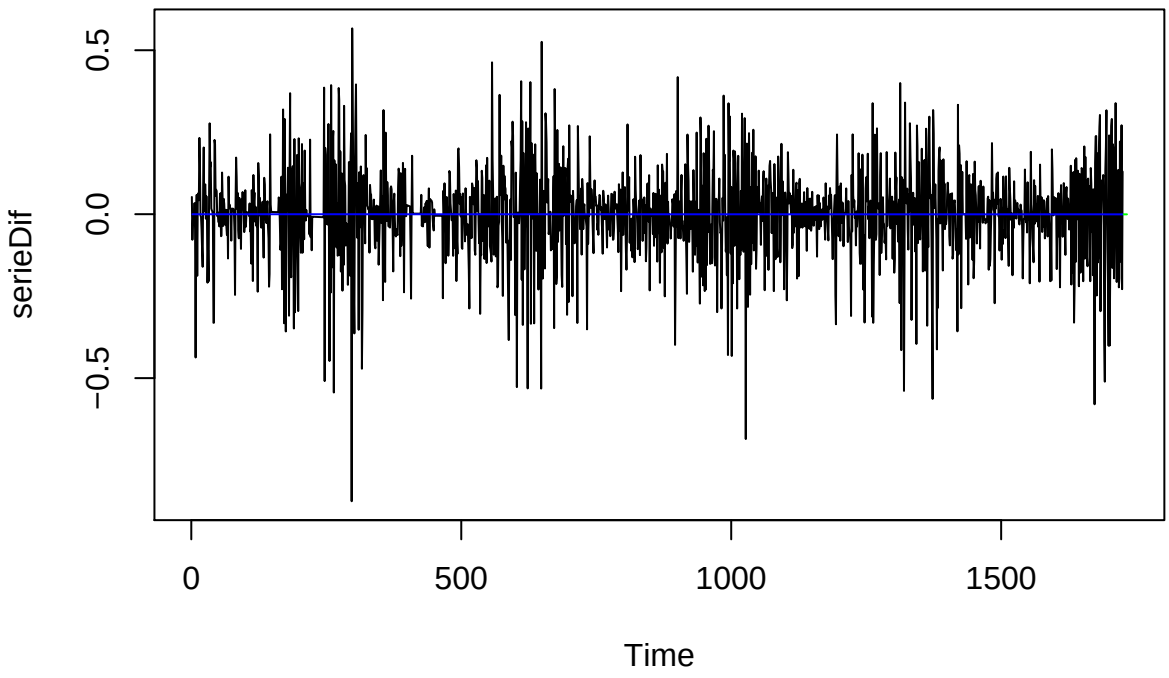
```
##
##  Jarque Bera Test
##
## data:  parametros.lm$residuals
## X-squared = 89.748, df = 2, p-value < 2.2e-16
```

Como esto tampoco funciona bien, probamos a realizar una serie de diferenciaciones a la serie para intentar suavizar la tendencia.

```
serieDif = diff(serie.train)
tiempoDif = tiempo.train[2:length(tiempo.train)]
tiempoPredDif = (tiempoDif[length(tiempoDif)]+1):(tiempoDif[length(tiempoDif)] + nPred)
parametros.lm = lm(serieDif ~ tiempoDif)

tendencia.train = parametros.lm$coefficients[1]+tiempoDif*parametros.lm$coefficients[2]
tendencia.pred = parametros.lm$coefficients[1]+tiempoPredDif*parametros.lm$coefficients[2]

plot.ts(serieDif, xlim=c(1, tiempoPredDif[length(tiempoPredDif)]))
lines(tiempoDif, tendencia.train, col="blue")
lines(tiempoPredDif, tendencia.pred, col="green")
```




```
JB.train = jarque.bera.test(parametros.lm$residuals)
JB.train
```

```
##
##  Jarque Bera Test
##
## data:  parametros.lm$residuals
## X-squared = 783.08, df = 2, p-value < 2.2e-16
```

Como el test de Jarque Bera sigue negando que los residuos sigan una distribución normal no podemos asumir que el modelo estime la tendencia correctamente, y además al ser esta tendencia calculada casi constante, no voy a eliminarla para los siguientes pasos.

El siguiente paso es eliminar la estacionalidad. En este caso como hemos dicho anteriormente, se entiende una estacionalidad de frecuencia 365 (anual contando en días).

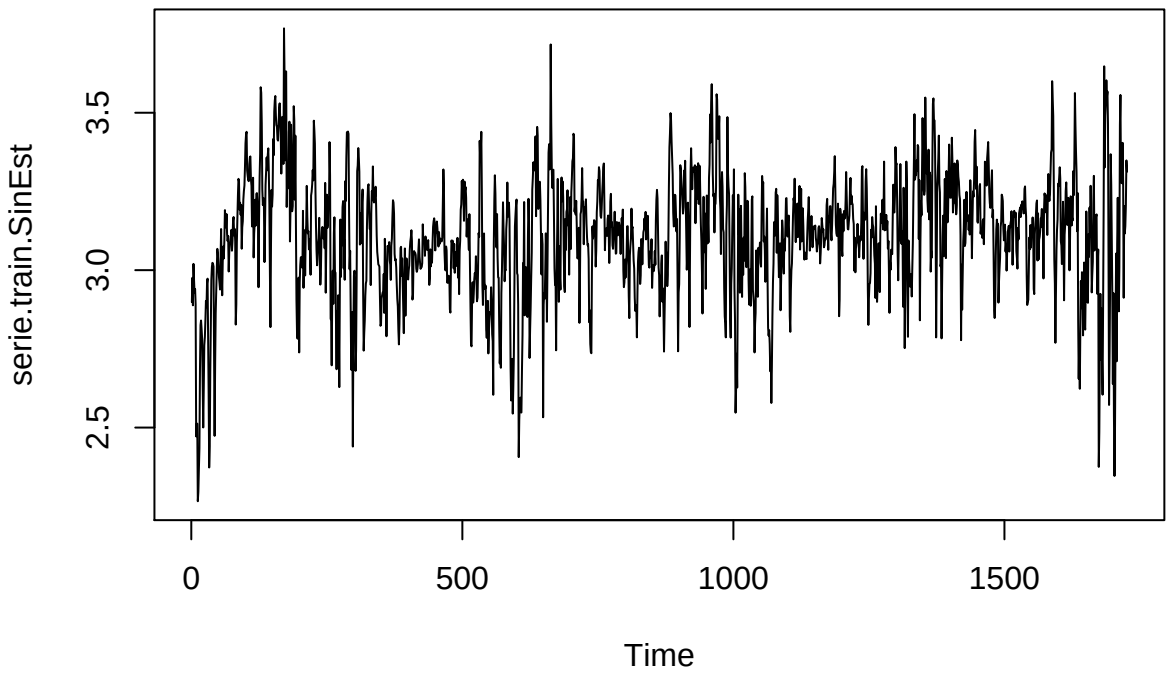
```
k = 365
estacionalidad = decompose(serie.ts.log)$seasonal[1:k]
```

Para restar esta estacionalidd hay que hacerlo de forma periódica a lo largo de toda la serie.

```
# Aprovecho el reciclaje de R para no tener que crear una variable auxiliar
serie.train.SinEst = serie.train - estacionalidad
```

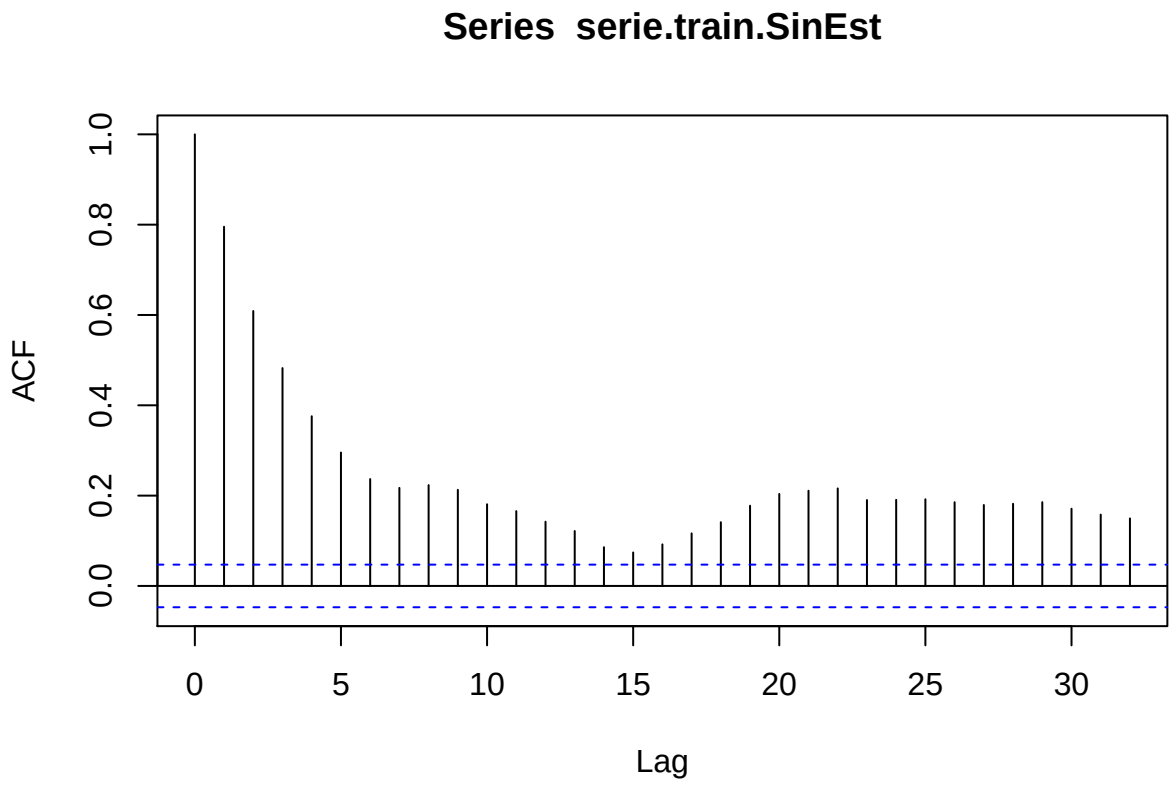
```
## Warning in serie.train - estacionalidad: longitud de objeto mayor no es
## múltiplo de la longitud de uno menor
```

```
plot.ts(serie.train.SinEst, xlim=c(1, tiempo.train[length(tiempo.train)]))
```

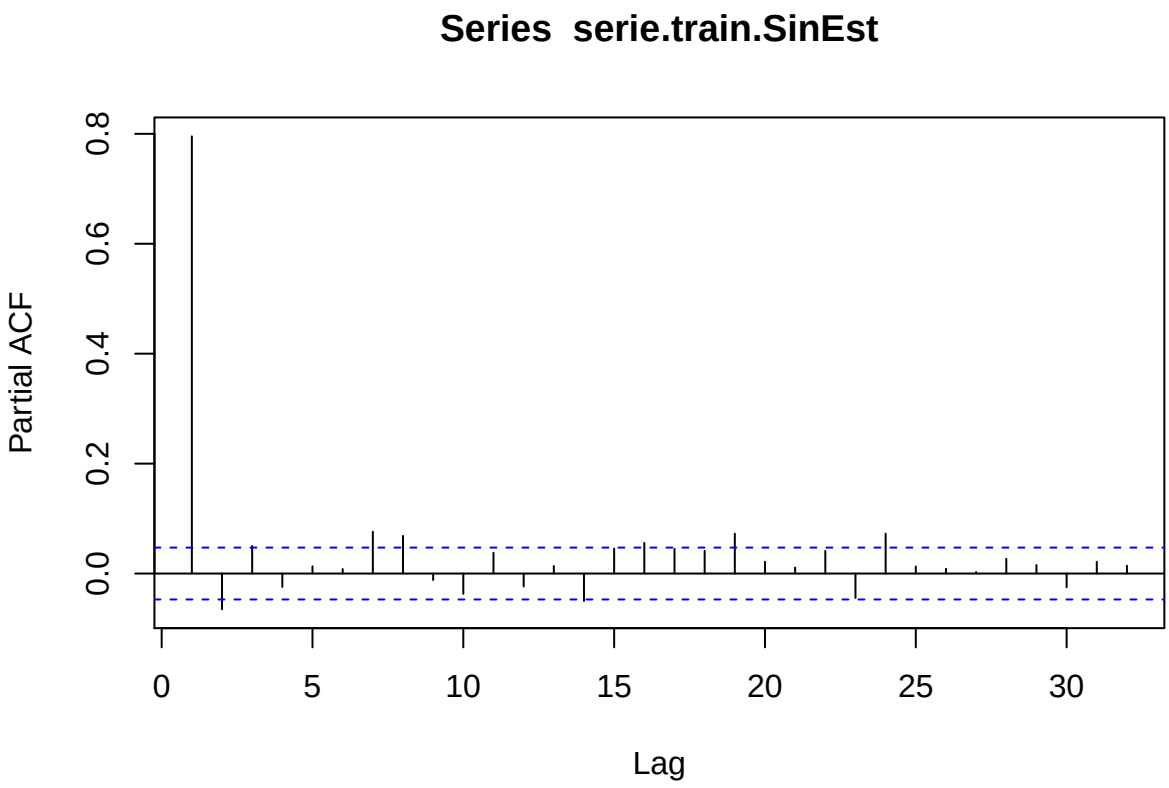



Ahora que la serie ya no tiene estacionalidad y hemos analizado que no teníamos que restarle la tendencia, vamos a comprobar si es estacionaria. Para ello podemos probar en principio a ejecutar un test ACF y un PACF.

```
acf(serie.train.SinEst)
```




```
pacf(serie.train.SinEst)
```




En el gráfico del test PACF se puede observar un rápido descenso, pero no se distingue un claro comportamiento de "olas". Por otro lado, en la gráfica del test ACF ni siquiera desciende rápido y no se queda por debajo del 0, por lo que es un comportamiento extraño. Para comprobar estadísticamente si se trata de una serie estacionaria podemos ejecutar el test de Dickey-Fuller aumentado y obtener nuestras conclusiones a partir de ello.

```
adf.test(serie.train.SinEst)
```

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

```
##
##  Augmented Dickey-Fuller Test
##
## data:  serie.train.SinEst
## Dickey-Fuller = -8.9979, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

Según el resultado arrojado por el test, el p-value bajo refleja una confianza alta en que la serie es estacionaria, por tanto se puede tratar de generar un modelo ARIMA. Al ser PACF el que parece que sigue una forma más de "olas", vamos a asumir que se trate de un modelo de medias móviles, cuyo grado podría ser cuestionable dado que todos los valores del test ACF se encuentran por encima del umbral. Voy a escoger el orden 3 porque es el último valor que se encuentra rondando el 0.5.

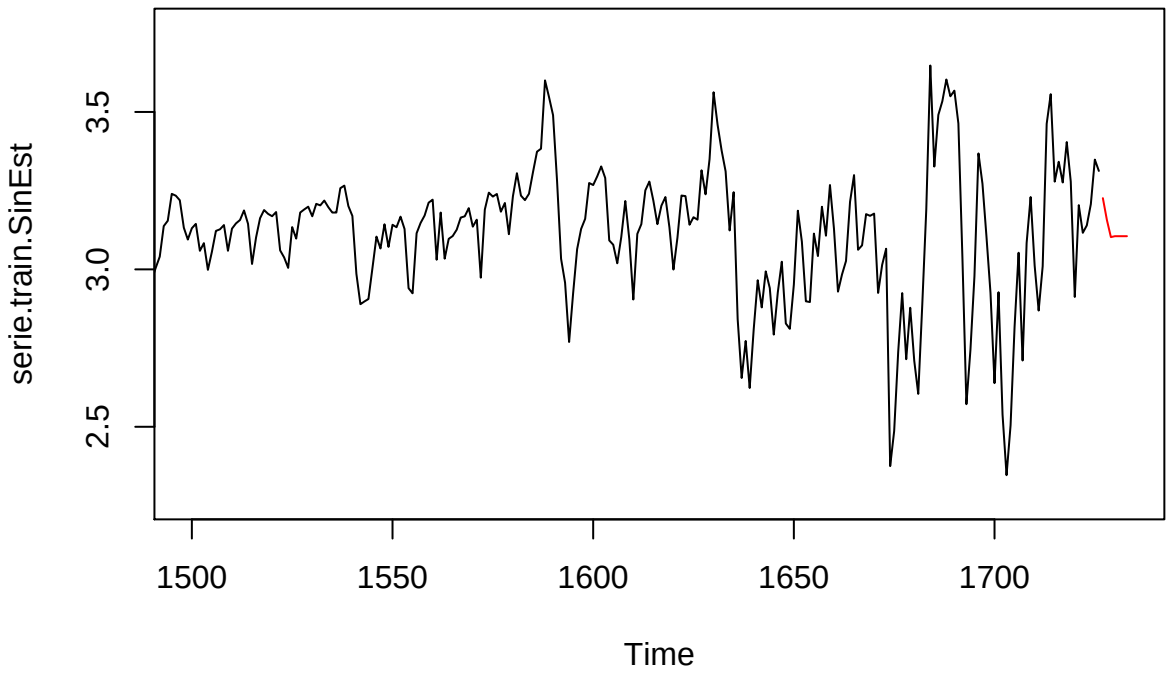
```
modelo = arima(serie.train.SinEst, order = c(0,0,3))
```

Una vez construido el modelo podemos generar las predicciones para los valores que buscamos (la primera semana de marzo).

```
predicciones = predict(modelo, n.ahead = nPred)
valoresPredichos = predicciones$pred
```

Podemos observar los valores predichos con el resto de los valores de la serie transformada. Voy a mostrar sólo los últimos datos para que se aprecien bien los valores calculados.

```
plot.ts(serie.train.SinEst, xlim=c(1500, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.pred, valoresPredichos, col="red")
```




Los datos parecen coherentes, pero vamos a aplicar el test de Box-Pierce para asegurarnos que los residuos resultantes del modelo sean aleatorios.

```
boxPierce.test = Box.test(modelo$residuals)
boxPierce.test
```

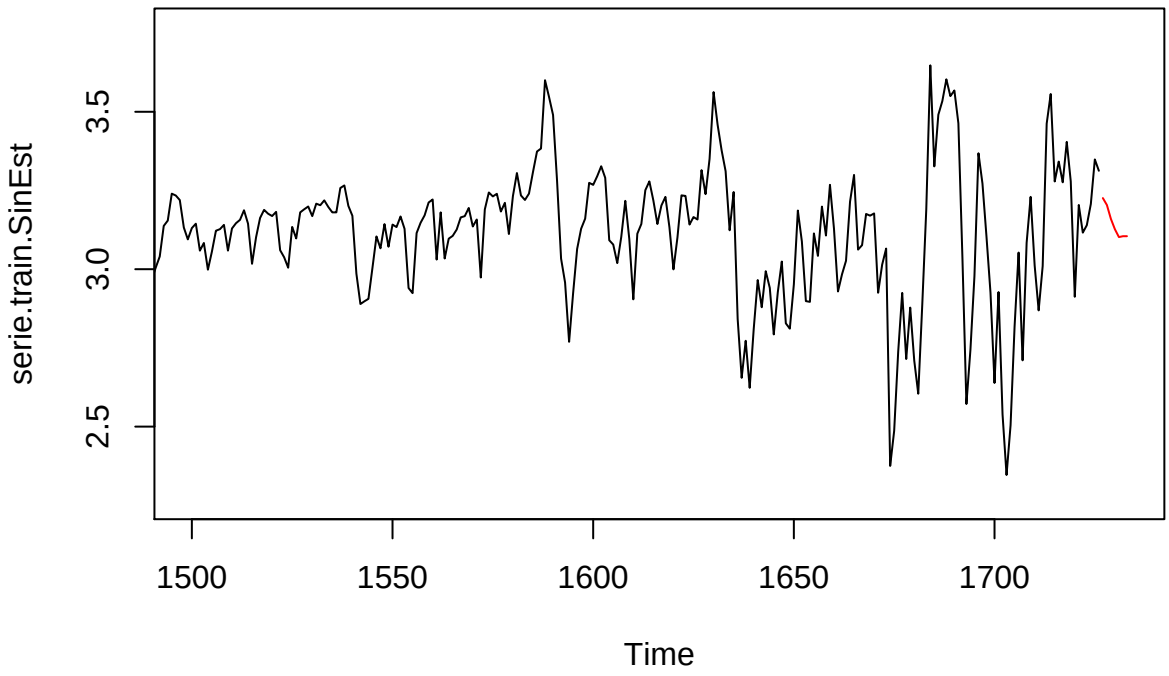
```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 3.9314, df = 1, p-value = 0.04739
```

El p-value del test de Box-Pierce nos hace pensar que los residuos no son aleatorios, por lo que hay una parte de la serie que se puede modelar de una forma mejor que el azar. Por ello, debido al amplio rango de datos que hay por encima del umbral para la gráfica de ACF, vamos a probar un nuevo modelo de medias móviles de rango mayor, en este caso de rango 5.

```
modelo = arima(serie.train.SinEst, order = c(0,0,5))
```

```
predicciones = predict(modelo, n.ahead = nPred)
valoresPredichos = predicciones$pred

plot.ts(serie.train.SinEst, xlim=c(1500, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.pred, valoresPredichos, col="red")
```




```
boxPierce.test = Box.test(modelo$residuals)
boxPierce.test
```

```
##
##  Box-Pierce test
##
## data:  modelo$residuals
## X-squared = 0.58608, df = 1, p-value = 0.4439
```

Este test ya sí nos asegura que los residuos son aleatorios, por lo que el modelo se adapta mejor a la serie. Con los tests de Jarque Bera y Shapiro-Wilk probamos si los residuos siguen una distribución normal.

```
jarqueBera.test = jarque.bera.test(modelo$residuals)
jarqueBera.test
```

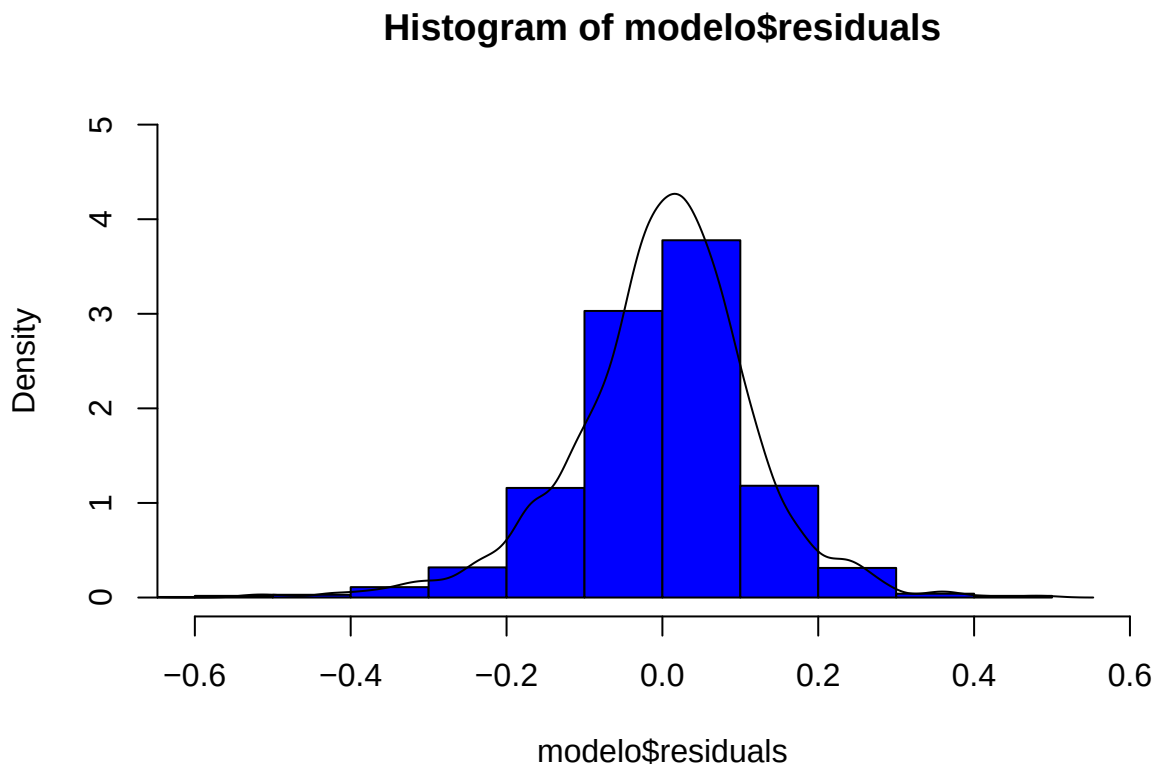
```
##
##  Jarque Bera Test
##
## data:  modelo$residuals
## X-squared = 455.61, df = 2, p-value < 2.2e-16
```

```
shapiroWilk.test = shapiro.test(modelo$residuals)
shapiroWilk.test
```

```
##
##  Shapiro-Wilk normality test
##
## data:  modelo$residuals
## W = 0.97124, p-value < 2.2e-16
```

Los tests de Jarque Bera y Shapiro-Wilk sin embargo dicen que los residuos no siguen una distribución normal. Aún así, si observamos su distribución en una gráfica mediante un histograma, podemos comprobar que se parece bastante.

```
hist(modelo$residuals, col="blue", prob=TRUE, ylim=c(0,5), xlim=c(-0.6,0.6))
lines(density(modelo$residuals))
```




Basándonos en esto y habiendo comprobado varias configuraciones de modelos ARIMA sin variar los resultados de este test, vamos a utilizar este modelo ARIMA8(0,0,5) para predecir los datos. A continuación hay que deshacer los cambios, que en este caso es simplemente sumar la estacionalidad y quitar la transformación logarítimca.

Como la serie comienza el 7 de mayo, para obtener la estacionalidad de los datos del 1 de marzo hay que restar al total de la estacionalidad de 365 los 7 días de mayo que ya hay, los 30 días de abril y los 31 días de marzo. Esto indica que la estacionalidad del 1 de marzo es el dato 297 del vector de estacionalidades, por lo que los 7 días de la semana a predecir ocuparían los puestos entre el 297 y el 303, incluidos.

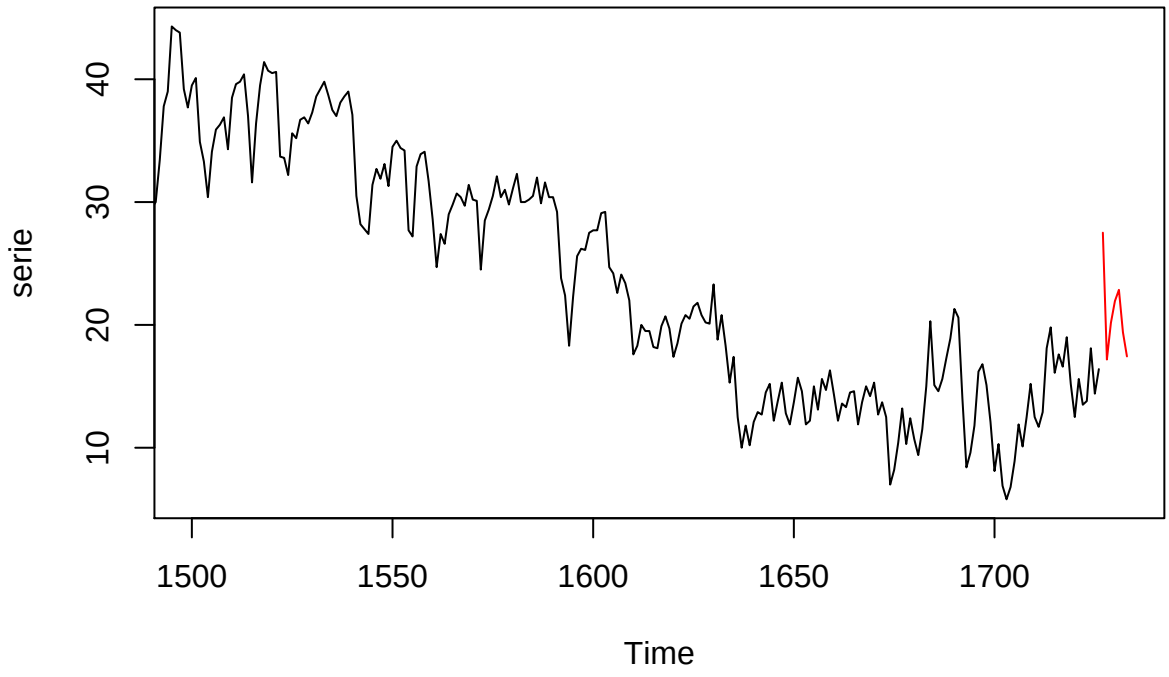
```
# Sumamos estacionalidad
# Los datos de la primera semana de marzo se corresponden
```

```r
# a los valores 297:303 del vector de estacionalidades
# de acuerdo a los cálculos anteriormente indicados
valoresPredichos.Est = valoresPredichos + estacionalidad[297:303]

# Deshacemos transformación logarítmica
valoresPredichos.EstExp = exp(valoresPredichos.Est)
```

Por último, mostramos los datos predichos junto a los que hemos utilizado para entrenar, de forma que continúen la serie y podamos comprobar cómo se adaptan. De nuevo incluimos solamente los últimos 300 datos aprox para que se pueda apreciar la predicción.

```r
plot.ts(serie, xlim=c(1500, tiempo.pred[length(tiempo.pred)]))
lines(tiempo.pred, valoresPredichos.EstExp, col="red")
```




Los valores predichos por el modelo han sido los siguientes.

```r
valoresPredichos.EstExp
```

```
## Time Series:
## Start = 1727
## End = 1733
## Frequency = 1
## [1] 27.50746 17.18056 20.15764 21.94198 22.85400 19.39961 17.43321
```