Trabajo Autónomo - Series Temporales

Néstor Rodriguez Vico

25 April, 2019

- Néstor Rodríguez Vico
- nrv23@correo.ugr.es

- Ejercicio de trabajo autónomo. Series temporales. Curso 2018-2019

Parte teórica.

Preprocesamiento.

El preprocesamiento aplicado ha sido la imputación de valores perdidos con el paquete *imputeTS* (https://cran.r-project.org/web/packages/imputeTS/index.html). Dentro de este paquete se ha usado la función *na.ma*, la cual hace una imputación mediante medias móviles ponderadas. Esta función utiliza un tamaño de ventana semi-adaptable para asegurar que todos los *NAs* son imputados.

Análisis de tendencia y de estacionalidad.

Para el estudio de la tendencia se han usado un conjunto de gráficos que nos permita visualizar la serie temporal. Uno de ellos es la descomposición de la serie mediante la función *decomponse*, la cual descompone la serie en sus tres componentes: estacionalidad, tendencia y ruido. De esta manera, podemos representar una serie temporal y_t cómo $y_t = S_t + T_t + R_t$, donde S_t representa la estacionalidad, T_t la tendencia y R_t el resto de la serie, lo cual se interpreta como el ruido de la misma.

En este trabajo se ha aplciado una descomposición aditiva, como la que podemos ver en las transparencias de teoría, la cual se hace mediante los siguientes pasos:

- Primero suponemos el periodo de la estacionalidad, el cual denotaremos por m. Si m es par, calcular la componente de la tendencia $\hat{T}_t$ usando un $2 \times m$-MA. Si m es impar, calcular la componente de la tendencia $\hat{T}_t$ usando un m-MA.
- Generamos una nueva serie sin tendencia como $y_t - \hat{T}_t$.
- Para calcular la componente de estacionalidad para cada mes, hacemos la media de la serie sin tendencia para dicho mes. Por ejemplo, la estacionalidad de Marzo es la media de los valores de la serie sin tendencia para el mes de Marzo. Una vez calculadas todas las componentes estacionales, se ajustan para asegurar que suman cero. La componente estacional se obtierne restrando los valores medios de cada mes a los datos para cada año de la serie. Esto nos da $\hat{S}_t$.
- La componente restante, $\hat{R}_t$, se calcula restando la tendencia y la componente estacional: $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$

Estacionareidad.

Tal y como hemos visto en clase de teoría, las series estacionarias son aquellas cuyas propiedades no dependen del tiempo. Para estudiar la estacionareidad podemos usar tests estadísticos, como puede ser usando el test de *Dickey-Fuller ampliado*. En mi caso particular, en las dos series estudiadas no pasaban el test y, por lo tanto, las series no eran estacionarias, incluso después de eliminar la tendencia y la estacionalidad. Para corregir esto, podemos diferenciar las series. Tras aplicar diferenciación, podemos volver a aplicar el test de

Dickey-Fuller ampliado, viendo ahora que el test si pasa para ambas series y, por lo tanto, podemos confirmar que estamos ante una serie estacionaria.

Otra técnica para detectar estacionaridad son los gráficos *ACF (Autocorrelation Function)*. Si la serie es estacionaria, los valores autocorrelados del *ACF* deben caer exponencialmente hacia cero, mientras que el *ACF* de una serie no estacionaria decaerá lentamente y tendrá patrones.

Modelado de series temporales.

Una vez tenemos una serie estacionaria podemos pasar al modelado de la serie. Para ello, vamos a hacer uso del gráfico *PACF (Partial Autocorrelations)*. El gráfico *ACF* muestra autocorrelaciones que miden la relación entre y_t y y_{t-k}. El problema que tiene dicho gráfico es que si los valores y_t y y_{t-1} están correlados y y_{t-1} y y_{t-2} también; y_t y y_{t-2} lo estarán, pero por influencia de y_{t-1}. *PACF* busca eliminar esta situación, representando la relación entre y_t y y_{t-k} tras eliminar los efectos de los valores retrasados. Este gráfico puede usarse para estimar los coeficientes en un modelo *AR* o para extraer el grado p en un modelo *ARIMA(p, d, q)*. Los modelos *ARIMA* son modelos que combinan modelos autoregresivos *AR*, diferenciación y modelos de medias móviles *MA*.

Los modelos autoregresivos, predicen una variable usando una combinación lineal de los valores anteriores de dicha variable, es decir, se aplica una regresión sobre la variable. Tal y como podemos ver en la transparencia 87, un modelo autoregresivo viene definido por la fórmula: $y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$, donde la última componente, ε_t, es el ruido blanco de la serie. El orden del modelo viene dado por p, el cual podemos obtener de los gráficos *PACF*. Este p es el último valor más autocorrelado con la variable estudiada.

Los modelos de medias móviles usan los errores de predicciones anteriores en lugar de los propios valores anteriores de la serie. Tal y como podemos ver en la transparencia 88, un modelo de medias móviles viene dado por la fórmula: $y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$, donde la última componente, ε_t, es el ruido blanco de la serie.

La combinación de ambos modelos da como resultado a los modelos *ARIMA*. Tal y como podemos ver en la transparencia 89, estos modelos vienen definidos por la fórmula: $y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$. Esta fórmula define un modelo *ARIMA(p, d, q)* donde:

- p es el orden de la parte autoregresiva.
- d es el grado de diferenciación (número de veces que se ha aplicado diferencición).
- q es el orden de la parte de medias móviles.

Parte práctica.

```
library(imputeTS)
library(forecast)
library(tseries)
library(xts)
```

Serie mensual.

Limpiamos el *environment* de trabajo:

```
rm(list = ls(all = T))
```

Se ha escogido la estación meteorológica con el identificador *5530E*, ubicada en el Aeropuerto de Granada.

```
serie <- read.csv("5530E.csv", sep = ";")
```

Dado que vamos a trabajar a nivel mensual, voy a suponer una frecuencia igual a 12, por el número de meses que hay en un año:

```
serie.ts <- ts(serie, frequency = 12)
```

Veamos si hay valores perdidos en la serie. Como vamos a trabajar con la temperatura máxima, vamos a centrarnos sólo en esta variable:

```
sum(is.na(serie$Tmax))
```

```
## [1] 124
```

Como podes ver, los hay, así que vamos a aplicar preprocesamiento.

Preprocesamiento

Para la imputación de valores perdidos se ha usado el paquete *imputeTs* (https://cran.r-project.org/web/packages/imputeTS/index.html):

```
serie <- imputeTS::na.ma(serie.ts)
serie.ts <- ts(serie, frequency = 12)
```

Veamos si sigue habiendo valores perdidos:

```
sum(is.na(serie))
```

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

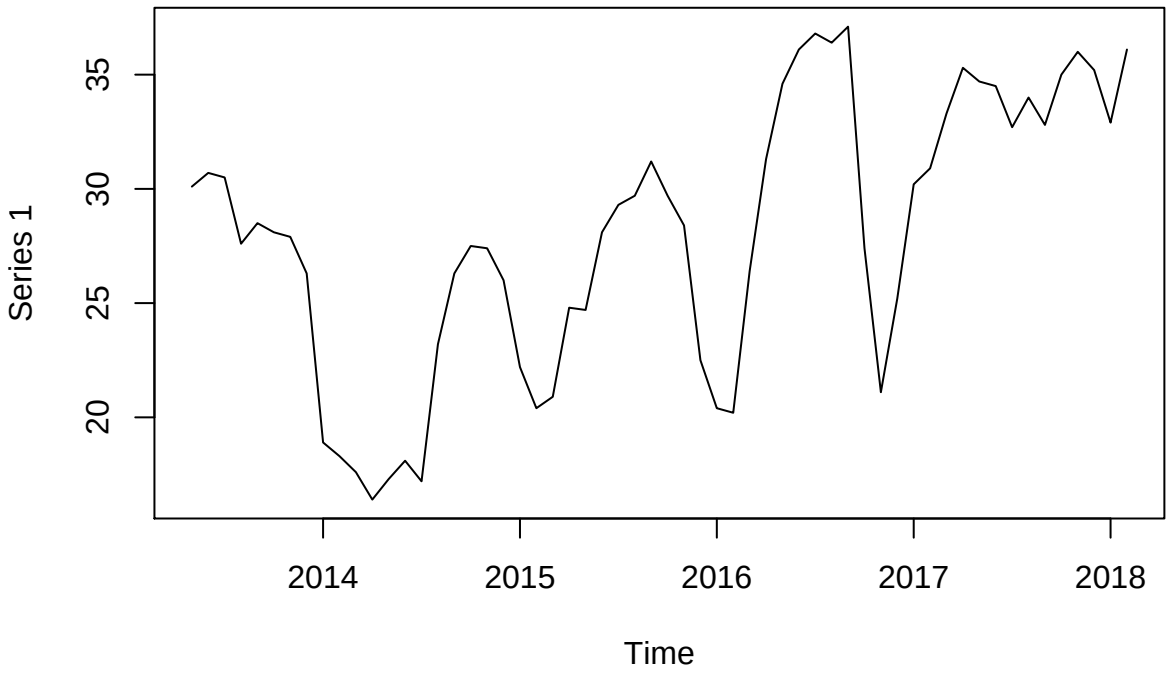
Como podemos ver, ya no hay valores perdidos en la serie.

Una vez que hemos retirado los valores perdidos, vamos a transformar la serie temporal para poder trabajar con ella a nivel mensual. Para ello, vamos a usar el paquete *xts* (https://cran.r-project.org/web/packages/xts/index.html)

```
# Agrupamos los datos por meses y nos quedamos sólo con la variable que
# representa la temeratura máxima, ya que es la que nos interesa
xts <- xts::xts(serie[,"Tmax"], order.by = as.Date(serie[,"Fecha"]), frequency = 1)
# El valor que vamos a tomar para un mes es la media de ese mes
serie <- xts::apply.monthly(xts, FUN = median)
# Convertimos los datos a una serie temporal
# Con los argumentos start = c(2013, 5) y end = c(2018, 2) indicamos que nuestra
# serie va a ser mensual. Esto lo podemos ver en los ejemplos de la documentación
# de la función ts: https://www.rdocumentation.org/packages/stats/versions/3.5.3/topics/ts
serie.ts <- ts(serie, frequency = 12, start = c(2013, 5), end = c(2018, 2))
```

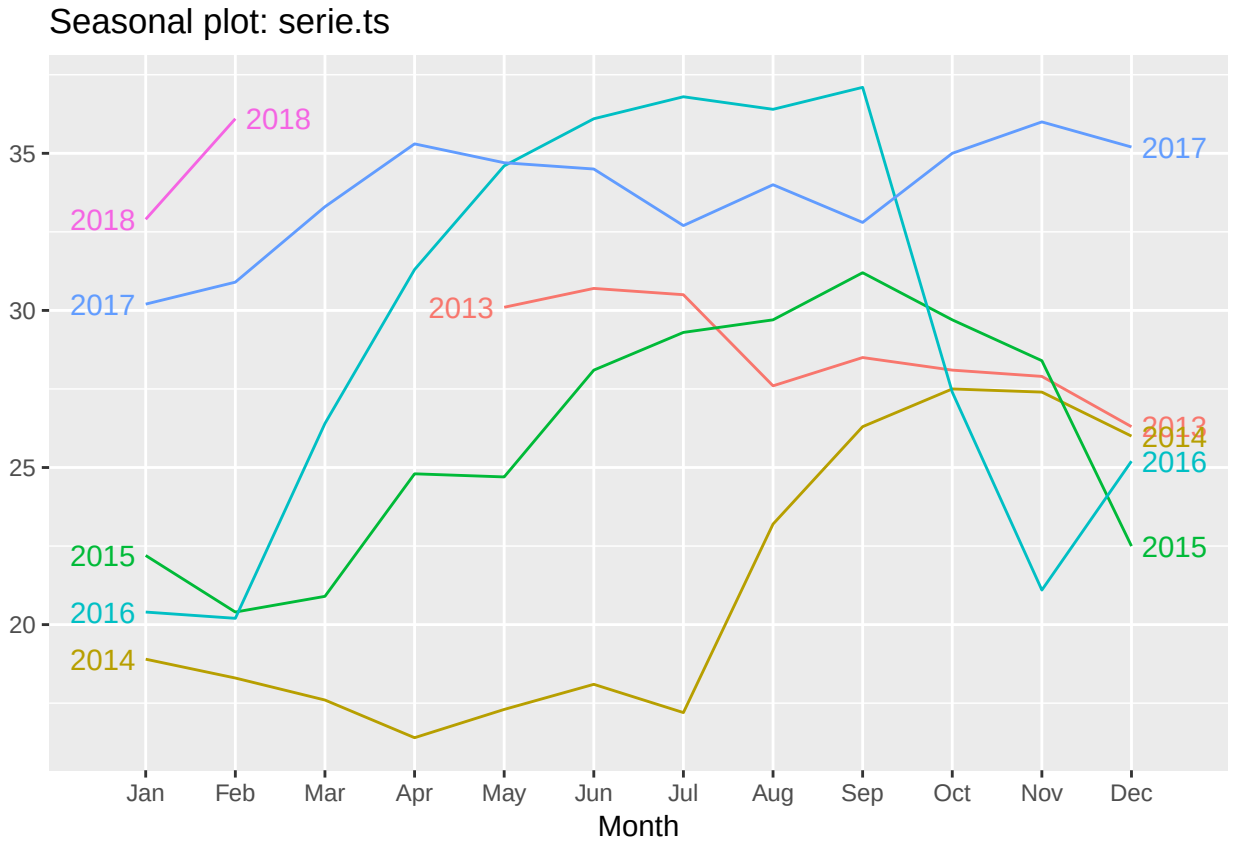
Análisis inicial de la serie.

```
plot.ts(serie.ts)
```



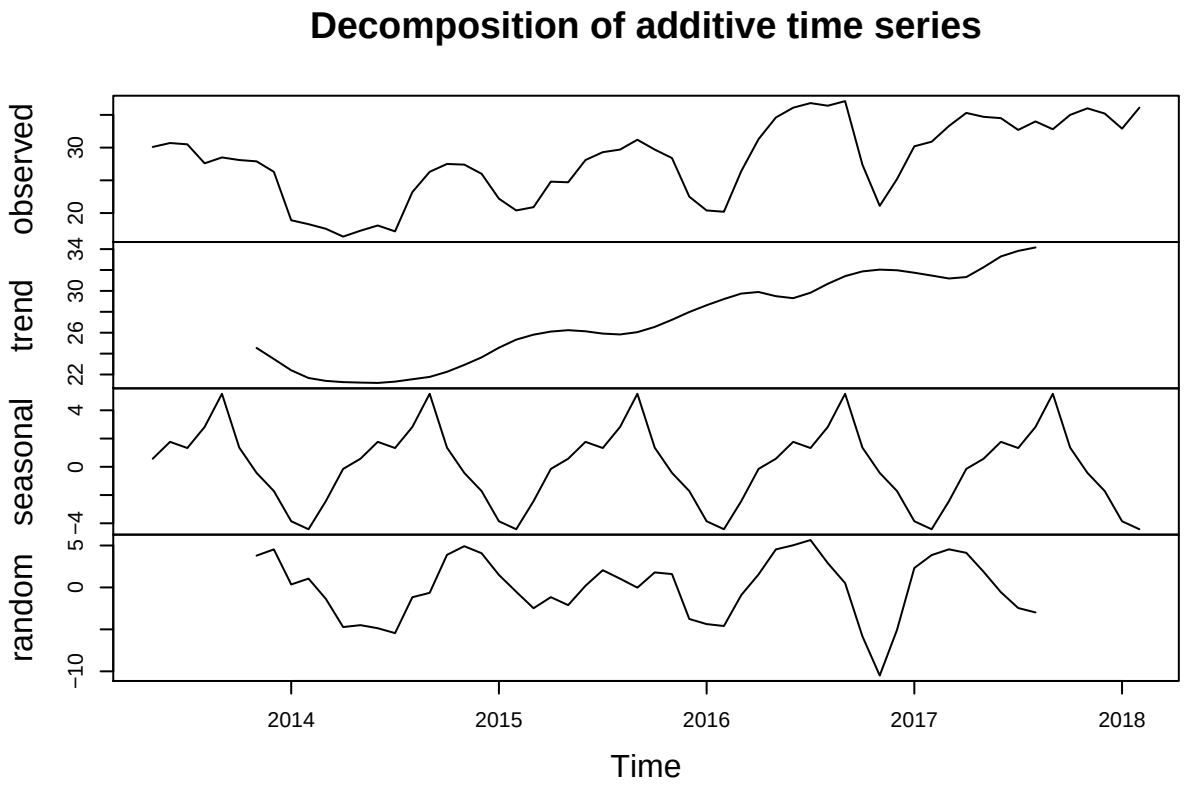
Como podemos ver, no hay ninguna tendencia en nuestra serie, ya que la temperatura no tiene a subir o a bajar a lo largo del tiempo, pero si se puede apreciar cierta estacionalidad con la que tendremos que lidiar. Al igual que en el trabajo guiado, podemos usar la función *ggseasonplot* del parquete *forecast* para ver más información acerca de la tendencia de la serie temporal (idea sacada de la página web https://otexts.com/fpp2/graphics.html):

```
forecast::ggseasonplot(serie.ts, year.labels = T, year.labels.left = T)
```



Analizando este gráfico podemos confirmar, una ves más, que no hay tendencia en la serie, ya que no se ve ni un crecimiento ni un decrecimiento de la temperatura según avanza el tiempo. Veamos la descomposición de la serie:

```
plot(decompose(serie.ts))
```

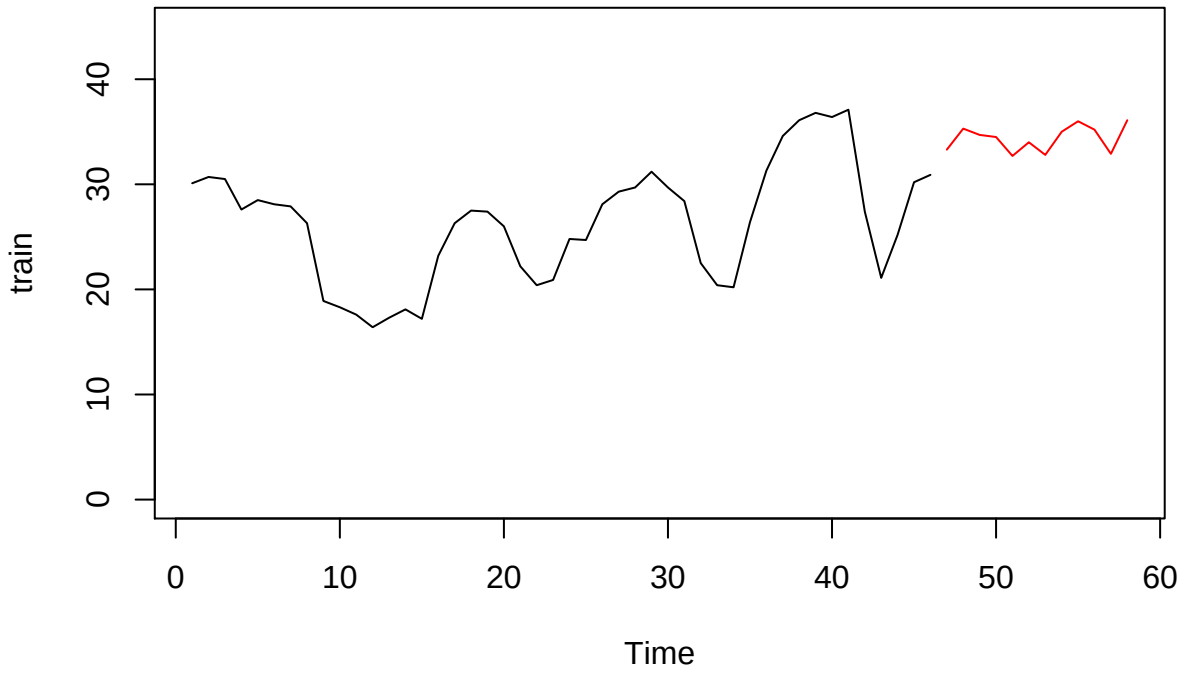


En este casi podemos ver que si hay cierta tendencia en los datos, así que vamos a intentar modelarla. También se puede observar que la estacionalidad no tiene tendencia (segunda gráfica) y, por lo tanto, no se ha aplicado ninguna transformación de los datos como se hizo en el trabajo guiado. Finalmente, podemos observar un fuerte patrón estacional (segunda gráfica).

Particionamiento en train y test.

Dado que estamos trabajando a nivel mensual, se ha seleccionado como conjunto de *test* la información del último año, es decir, los últimos dos meses disponible y el resto como conjunto de *train*.

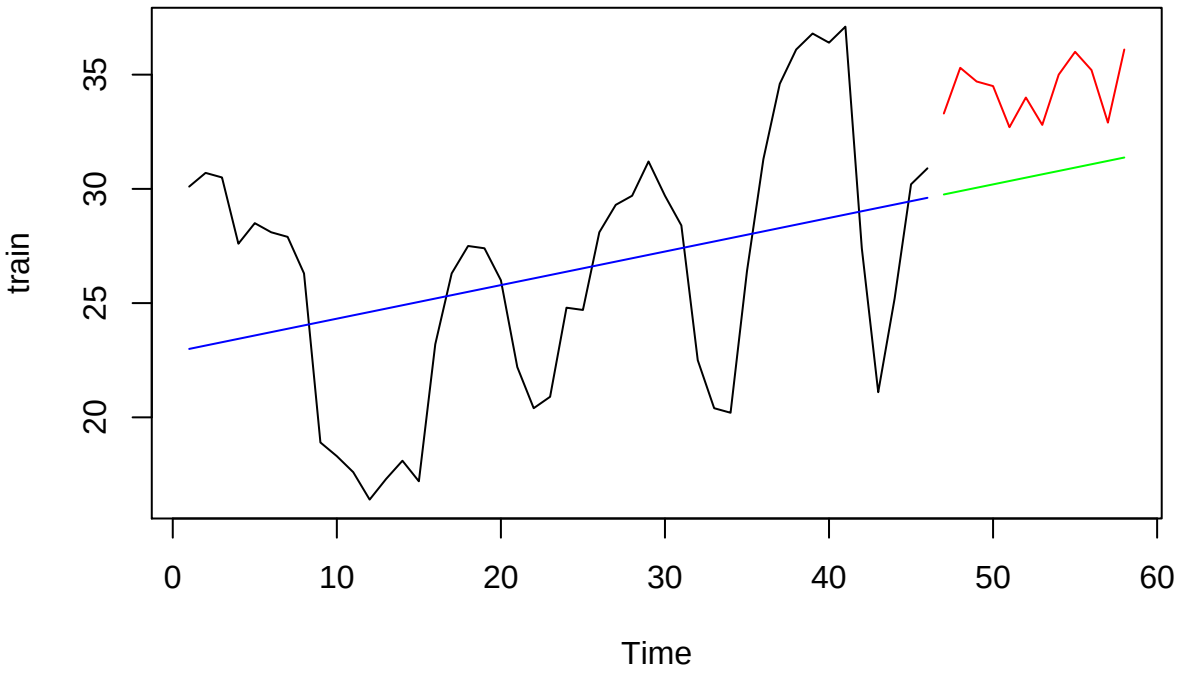
```
n.test <- 12
train <- serie.ts[1:(length(serie.ts) - n.test)]
tiempo.train <- 1:length(train)
test <- serie.ts[(length(serie.ts) - n.test + 1):length(serie.ts)]
tiempo.test <- (tiempo.train[length(tiempo.train)] + 1):
  (tiempo.train[length(tiempo.train)] + n.test)

plot.ts(train, xlim = c(1, tiempo.test[length(tiempo.test)]), ylim=c(0, 45))
lines(tiempo.test, test, col = "red")
```



Análisis de Tendencia.

A continuación, modelaremos la tendencia. Debemos modelar primero la tendencia ya que puede darse el caso de que no seamos capaces de detectar estacionalidad si no quitamos primero la tendencia. Para modelar la tendencia vamos a usar una aproximaxión funcional. Asumiremos un comportamiento lineal en este caso como hipótesis.

```
parametros.H1 <- lm(train ~ tiempo.train)

#Calculamos la estimación de la tendencia.
TendEstimadaTr.H1 <- parametros.H1$coefficients[1]+tiempo.train*parametros.H1$coefficients[2]

TendEstimadaTs.H1 <- parametros.H1$coefficients[1]+tiempo.test*parametros.H1$coefficients[2]

#Mostramos en la misma figura la serie y la tendencia estimada
plot.ts(train, xlim=c(1, tiempo.test[length(tiempo.test)]))
lines(tiempo.train, TendEstimadaTr.H1, col="blue")
lines(tiempo.test, test, col="red")
lines(tiempo.test, TendEstimadaTs.H1, col="green")
```



Comprobamos que la hipótesis de tendencia lineal es válida. Para ello se aplica un T-test, asumiendo normalidad en los datos (u otro test no paramétrico si los datos no son normales), que compare los residuos del ajuste con los errores del modelo en test. En nuestro caso, todos los tests de normalidad dan p-value>0.05. Asumimos normalidad. También el T-test da un p-value>0.05. No existen diferencias significativas en los datos.

```
#Test de normalidad de Jarque Bera
jarque.bera.test(parametros.H1$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  parametros.H1$residuals
## X-squared = 2.9379, df = 2, p-value = 0.2302
```

```
jarque.bera.test((TendEstimadaTs.H1 - test))
```

```
##
##  Jarque Bera Test
##
## data:  (TendEstimadaTs.H1 - test)
## X-squared = 1.0139, df = 2, p-value = 0.6023
```

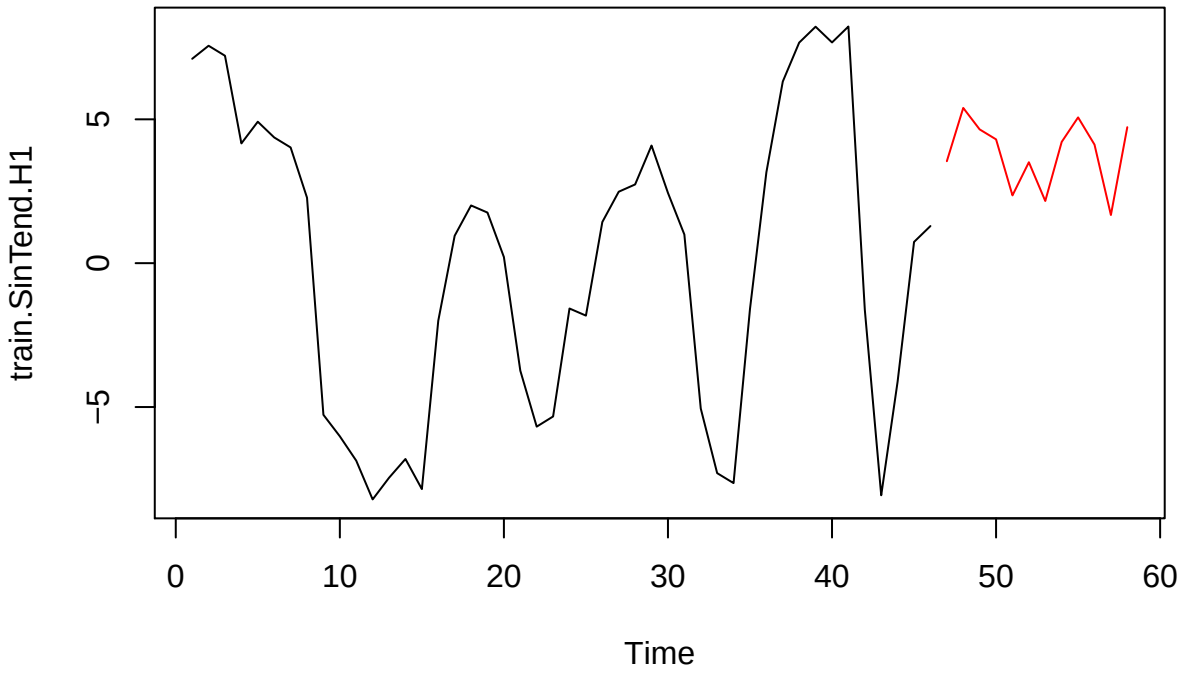
```
#Test de Student
t.test(c(parametros.H1$residuals, TendEstimadaTs.H1-test))
```

```
##
##  One Sample t-test
##
## data:  c(parametros.H1$residuals, TendEstimadaTs.H1 - test)
## t = -1.212, df = 57, p-value = 0.2305
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -2.0910236  0.5141702
## sample estimates:
```

```
##  mean of x
## -0.7884267
```

Como podemos ver, no se encuentran diferencias significativas tanto en el proceso de entrenamiento (p-value 0.4158) como en el proceso de test (0.6442), así que podemos decir que una aproximación con un modelo lineal es interesante. El test de Student no muestra diferenias significativas, así que podemos confirmar que el modelo lineal puede ser usado. A continuación eliminamos la tendencia en la serie:

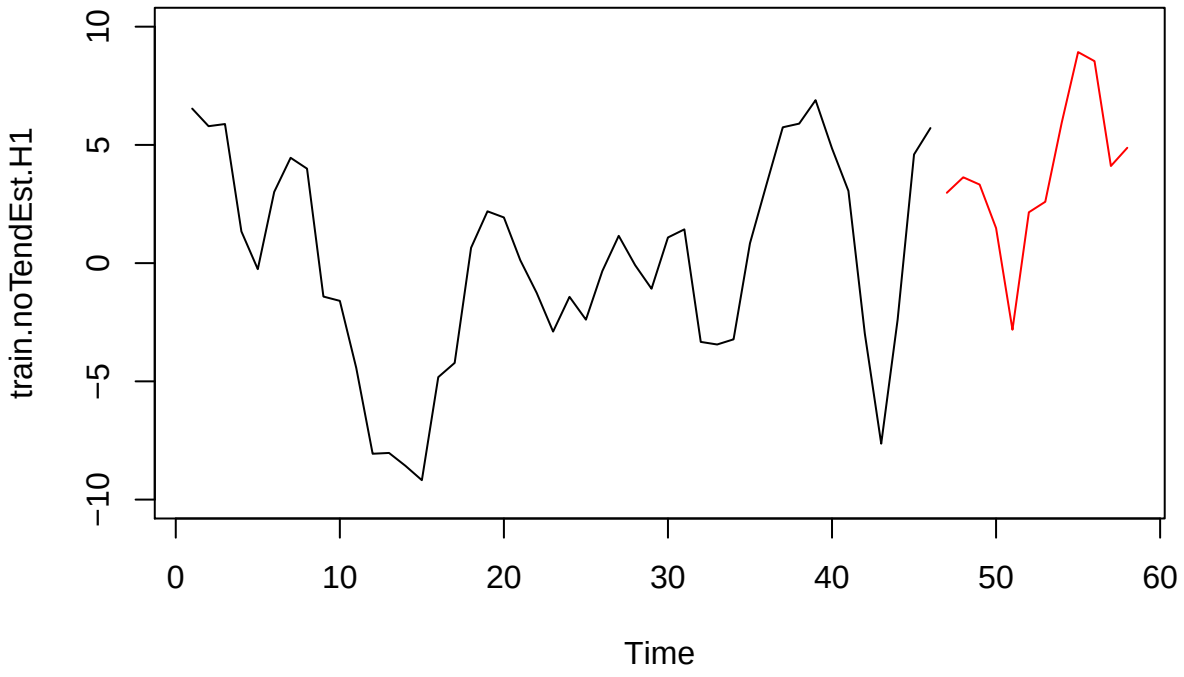
```
train.SinTend.H1 <- train - TendEstimadaTr.H1
test.SinTend.H1 <- test - TendEstimadaTs.H1
plot.ts(train.SinTend.H1, xlim = c(1, tiempo.test[length(tiempo.test)]))
lines(tiempo.test, test.SinTend.H1, col = "red")
```



Análisis de Estacionalidad.

El siguiente paso es eliminar la estacionalidad. Para ello, vamos a usar un modelo de diferenciación. Para eliminar la estacionalidad, podemos hacer uso de las salidas de la función "decompose":

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

# Eliminamos estacionalidad para el modelo
aux <- rep(estacionalidad, length(train) / length(estacionalidad))

train.noTendEst.H1 <- train.SinTend.H1 - aux
test.noTendEst.H1 <- test.SinTend.H1 - estacionalidad

plot.ts(train.noTendEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(-10, 10))
lines(tiempo.test, test.noTendEst.H1, col = "red")
```



Análisis de Estacionariedad.

Con la serie sin tendencia ni estacionalidad, debemos comprobar si es estacionaria antes de hipotetizar modelos de predicción. Usamos el test de Dickey-Fuller aumentado.

```
# Comprobamos el test de Dickye-Fuller aumentado para la estacionaridad
adf <- adf.test(train.noTendEst.H1)
adf
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  train.noTendEst.H1
## Dickey-Fuller = -2.7586, Lag order = 3, p-value = 0.2723
## alternative hypothesis: stationary
```

Al no superarse el test (obtenemos un *p-value* de 0.2723395, el cual es mayor que 0.05) podemos decir que la serie no es estacionaria. Para solucionar esto, diferenciamos la serie:

```
# Diferenciamos la serie
train.noTendEstDiff.H1 <- diff(train.noTendEst.H1)
test.noTendEstDiff.H1 <- diff(test.noTendEst.H1)
```

Una vez diferenciada, volvemos a aplicar el test:

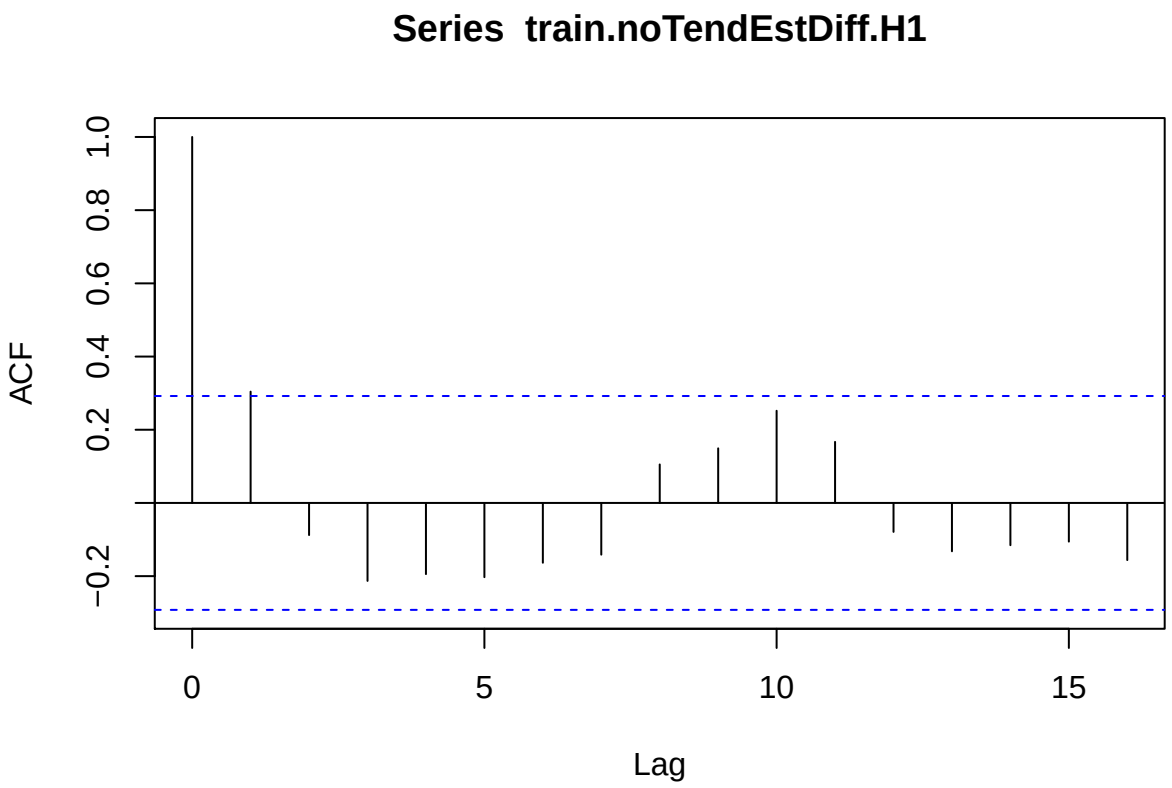
```
# Volvemos a aplicar el test
adf <- adf.test(train.noTendEstDiff.H1)
adf
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  train.noTendEstDiff.H1
## Dickey-Fuller = -3.8326, Lag order = 3, p-value = 0.02528
## alternative hypothesis: stationary
```

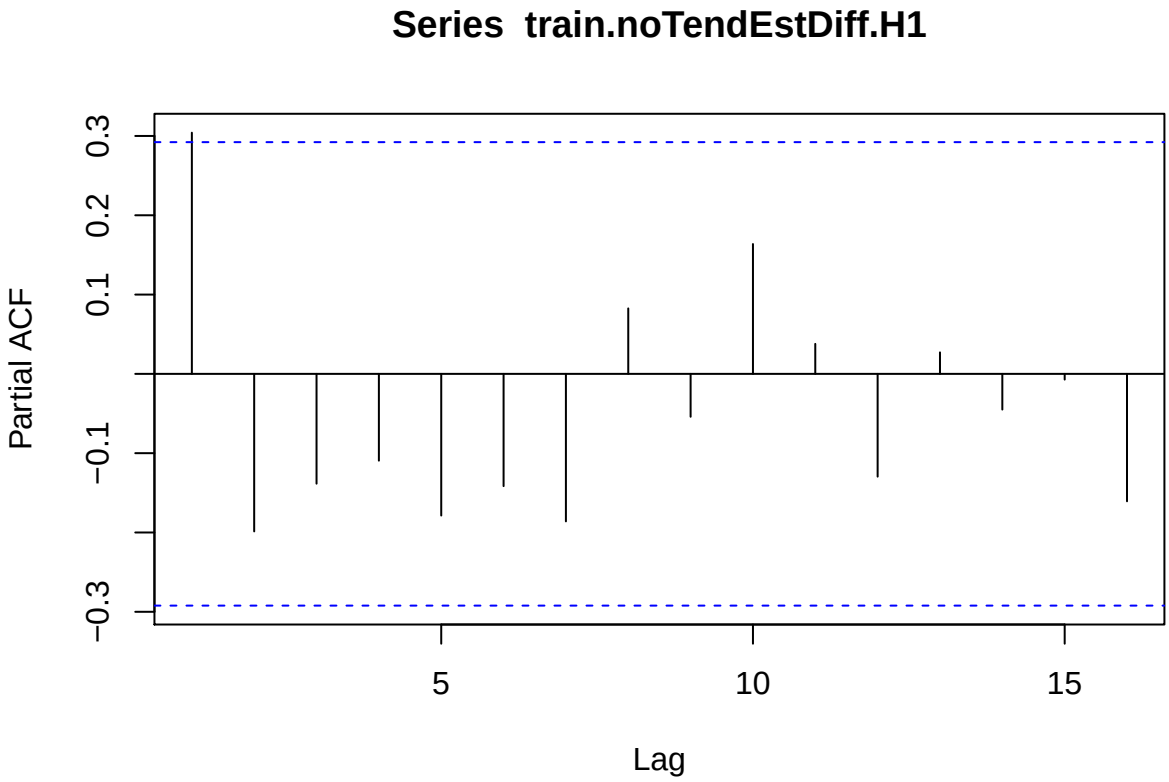
Podemos ver que, tras aplicar diferenciación, la serie que tenemos si es estacionaria (ya que obtiene un *p-value* de 0.0252757, el cual es menor que 0.05).

Finalmente, podemos comprobar que la serie es estacionaria vamos a visualizar el gráfico ACF y el gráfico PACF:

```
acf(train.noTendEstDiff.H1)
```




```
pacf(train.noTendEstDiff.H1)
```



Selección del modelo de predicción.

Viendo los gráficos ACF y PACF podemos de decir son los gráficos típicos de un modelo autoregresivo. Para ver de que orden, sólo debemos coger el índice del último *lag* que pasa los umbrales en el gráfico PACF. En mi caso, nos encontramos ante un modelo autoregresivo de orden 1, es decir, un *AR(1)*. Cómo hemos diferenciado una vez, podríamos incluir la diferencia dentro del modelo ajustando un *ARIMA(1, 1, 0)* sobre la serie sin tendencia y sin estacionalidad.

```
# Ajustamos el modelo
arima1.1.0.H1 <- arima(train.noTendEst.H1, order = c(1, 1, 1))
```

Validación del modelo de predicción.

Para validad el modelo usado para le predicción voy calcular las los valores de la serie temporal para el conjunto de test y voy a calcular el error cuadrático acumulado del ajuste, tanto en el proceso de entrenamiento como en el proceso de test:

```
ajustados.arima1.1.0.H1 <- train.noTendEst.H1 + arima1.1.0.H1$residuals

# Calculamos las predicciones
pred.arima1.1.0.H1 <- predict(arima1.1.0.H1, n.ahead = n.test)$pred

# Calculamos el error cuadrático acumulado del ajuste, en ajuste y en test
error.train.arima1.1.0.H1 <- sum((arima1.1.0.H1$residuals)^2)
cat("error.train.arima1.1.0.H1:", error.train.arima1.1.0.H1, "\n")
```

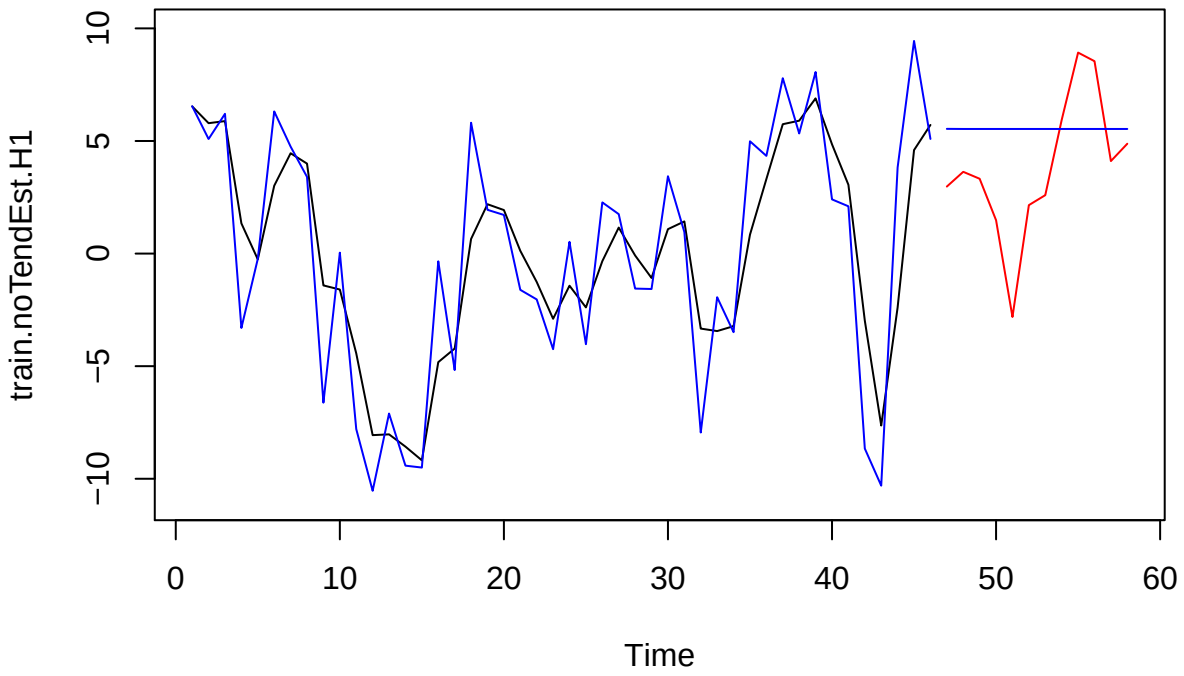
```
## error.train.arima1.1.0.H1: 313.8306
```

```
error.test.arima1.1.0.H1 <- sum((pred.arima1.1.0.H1-test.noTendEst.H1)^2)
cat("error.test.arima1.1.0.H1:", error.test.arima1.1.0.H1, "\n")
```

```
## error.test.arima1.1.0.H1: 144.4696
```

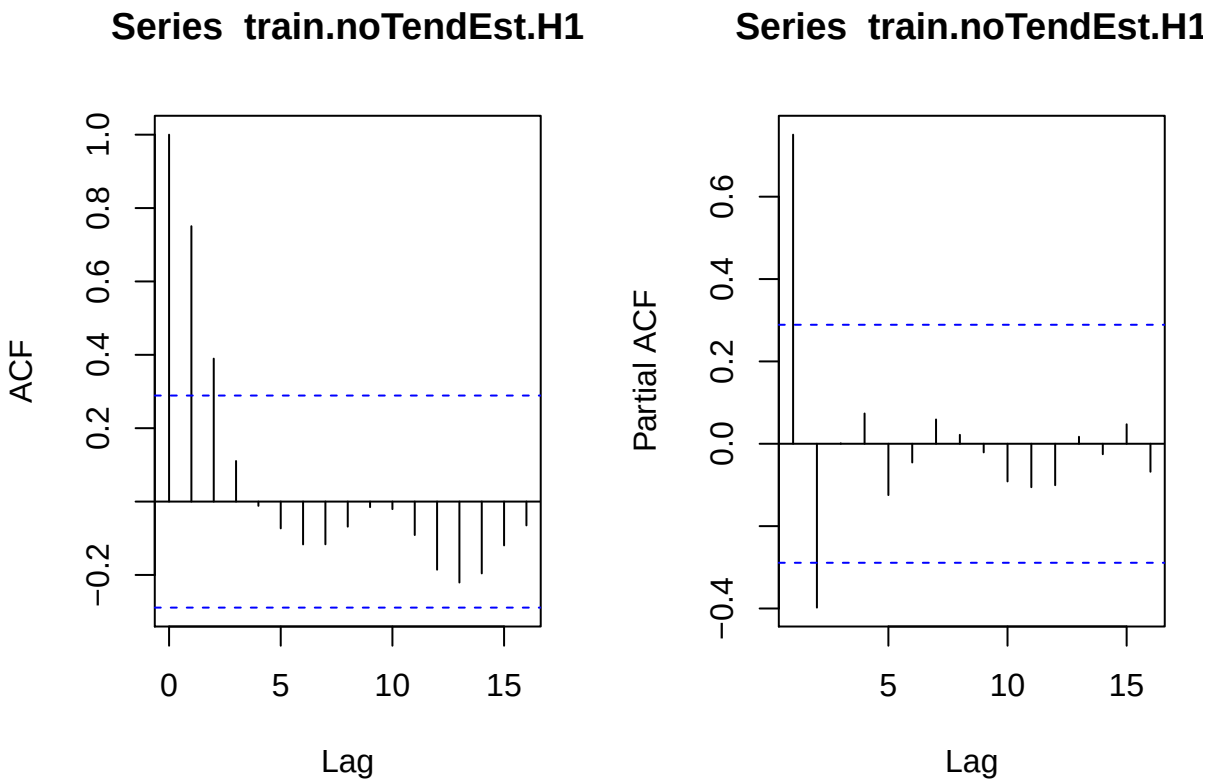
También voy a mostrar las gráficas del ajuste y predicción en test:

```
plot.ts(train.noTendEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(-11, 10))
lines(ajustados.arima1.1.0.H1, col = "blue")
lines(tiempo.test, test.noTendEst.H1, col = "red")
lines(tiempo.test, pred.arima1.1.0.H1, col = "blue")
```



Si nos fijamos en la como nuestro modelo predice el conjunto de test, podríamos decir que no lo hace del todo bien. Si observamos de nuevo el gráfico ACF y PACF de la serie que acabamos de modelar con nuestro modelo ARIMA:

```
par(mfrow=c(1,2))
acf(train.noTendEst.H1)
pacf(train.noTendEst.H1)
```




```
par(mfrow=c(1,2))
```

En el gráfico ACF podemos observar que el dato importante para predicir un dato es el propio dato. En el gráfico PACF podemos ver que sólo el *lag* 2 supera el umbral, esto quiere decir que para predecir un dato, necesitamos ese dato y los dos anteriores. Aún así, supera el umbral por muy poco. Estos dos gráficos nos indican que estamos ante una serie que es ruido blanco.

Por lo tanto, vamos a validar el modelo aplicando los siguientes tests estadísticos:

- Test de Box-Pierce para aleatoriedad de residuos.

```
#Tests para la seccion del modelo y su validacion
box.pierce <- Box.test(arima1.1.0.H1$residuals)
box.pierce
```

```
##
##  Box-Pierce test
##
## data:  arima1.1.0.H1$residuals
## X-squared = 0.0027687, df = 1, p-value = 0.958
```

Obtenemos un *p-value* de 0.9580361, lo cual nos indica que el test se pasa y, por lo tanto, los errores son aleatorios. Esto nos indica que nuestro modelo modela todo lo que puede modelar.

- Tests de Jarque Bera y Shapiro-Wilk para normalidad de residuos.

```
#Test de normalidad de Jarque Bera
jbt <- jarque.bera.test(arima1.1.0.H1$residuals)
jbt
```

```
##
##  Jarque Bera Test
##
```

```
## data:  arima1.1.0.H1$residuals
## X-squared = 0.22824, df = 2, p-value = 0.8922
```

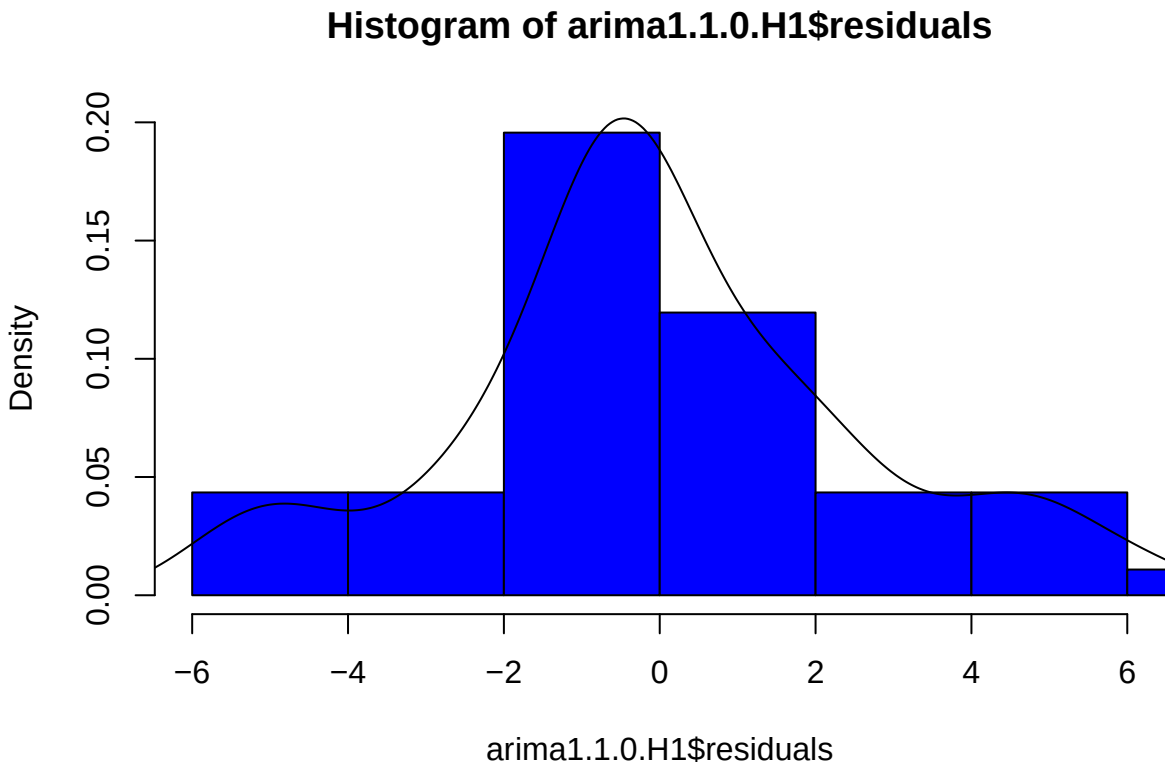
```
#Test de normalidad de Shaphiro-Wilk
saphiro <- shapiro.test(arima1.1.0.H1$residuals)
saphiro
```

```
##
##  Shapiro-Wilk normality test
##
## data:  arima1.1.0.H1$residuals
## W = 0.97165, p-value = 0.3187
```

Obtenemos un *p-value* de 0.8921527 y 0.3186962 respectivamente, por lo tanto, ambos tests pasan. Esto nos indica que los residuos son normales.

- Mostramos un histograma y la función de densidad para confirmación gráfica.

```
hist(arima1.1.0.H1$residuals, col = "blue", prob = T, ylim=c(0, 0.2), xlim=c(-6, 6))
lines(density(arima1.1.0.H1$residuals))
```



Comparación con otros modelos.

Al igual que hemos probado un modelo *ARIMA(1, 1, 0)*, voy a probar distintos modelos. Para comparar los distintos modelos se va a usar el valor del error cuadrático acumulado. Otra opción de métrica estudiada en la asignatura para comprar los modelos es el *AIC*. El problema que tiene el *AIC* es que es una métrica que se obtiene sólo en base a como funciona el modelo para los datos empleados en el proceso de entrenamiento. Sin embargo, el error cuadrático acumulado permite saber como se comporta el modelo para un conjunto de test el cual nunca ha visto.

ARIMA(2, 2, 0).

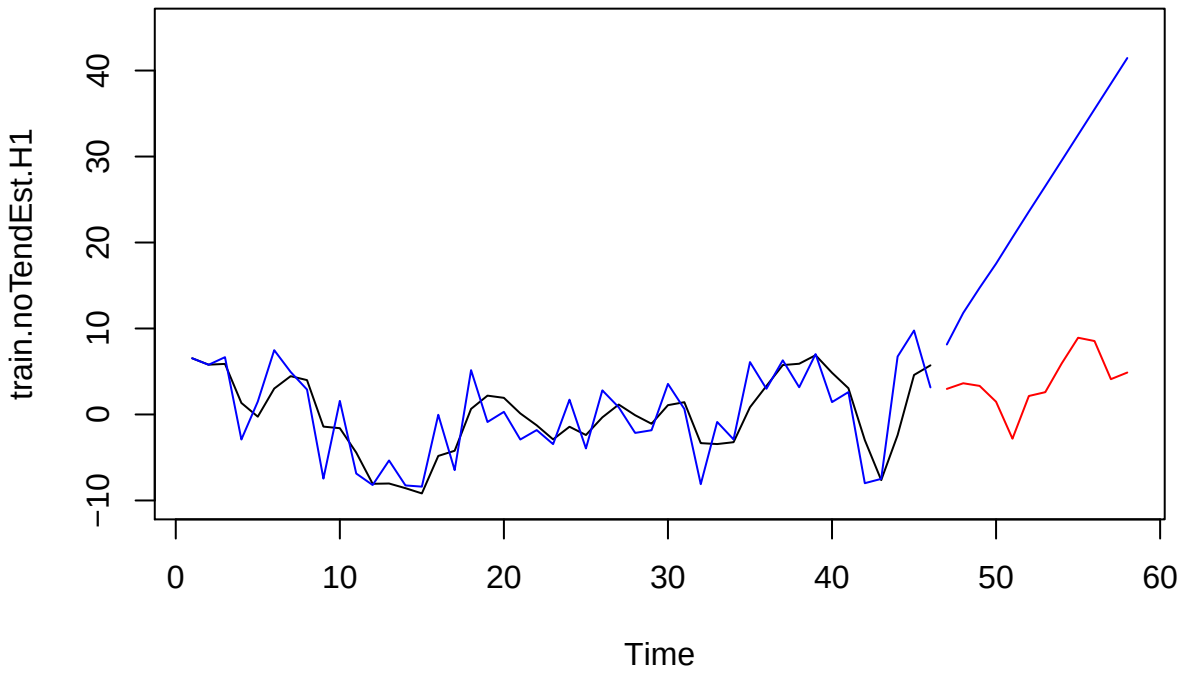
```
# Ajustamos el modelo
arima2.2.0.H1 <- arima(train.noTendEst.H1, order = c(2, 2, 0))

ajustados.arima2.2.0.H1 <- train.noTendEst.H1 + arima2.2.0.H1$residuals

# Calculamos las predicciones
pred.arima2.2.0.H1 <- predict(arima2.2.0.H1, n.ahead = n.test)$pred

# Calculamos el error cuadrático acumulado del ajuste, en ajuste y en test
error.train.arima2.2.0.H1 <- sum((arima2.2.0.H1$residuals)^2)
cat("error.train.arima2.2.0.H1:", error.train.arima2.2.0.H1, "\n")
```

```
## error.train.arima2.2.0.H1: 426.3917
```

```
error.test.arima2.2.0.H1 <- sum((pred.arima2.2.0.H1-test.noTendEst.H1)^2)
cat("error.test.arima2.2.0.H1:", error.test.arima2.2.0.H1, "\n")
```

```
## error.test.arima2.2.0.H1: 6422.443
```

```
plot.ts(train.noTendEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(-10, 45))
lines(ajustados.arima2.2.0.H1, col = "blue")
lines(tiempo.test, test.noTendEst.H1, col = "red")
lines(tiempo.test, pred.arima2.2.0.H1, col = "blue")
```



ARIMA(0, 0, 1).

Una prueba que no hemos hecho ha sido la de probar un modelo de medias móviles solo. Vamos allá:

```
# Ajustamos el modelo
arima0.0.1.H1 <- arima(train.noTendEst.H1, order = c(0, 0, 1))

ajustados.arima0.0.1.H1 <- train.noTendEst.H1 + arima0.0.1.H1$residuals
```

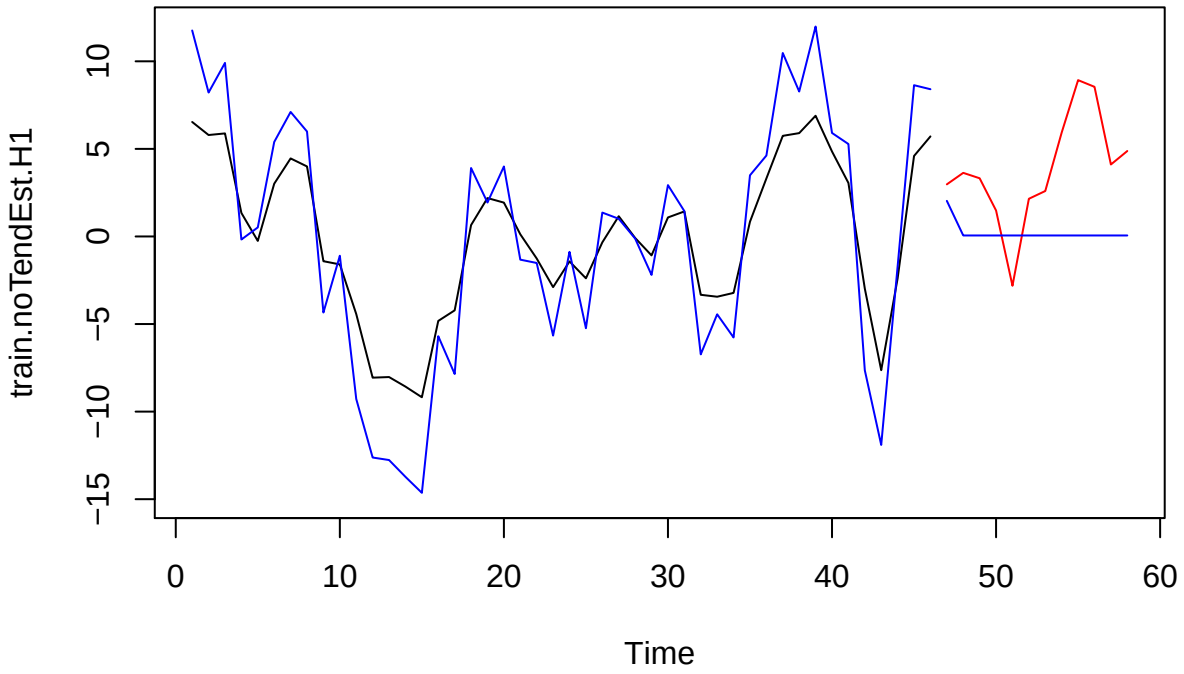
```r
# Calculamos las predicciones
pred.arima0.0.1.H1 <- predict(arima0.0.1.H1, n.ahead = n.test)$pred

# Calculamos el error cuadrático acumulado del ajuste, en ajuste y en test
error.train.arima0.0.1.H1 <- sum((arima0.0.1.H1$residuals)^2)
cat("error.train.arima0.0.1.H1:", error.train.arima0.0.1.H1, "\n")
```

```
## error.train.arima0.0.1.H1: 407.3635
```

```r
error.test.arima0.0.1.H1 <- sum((pred.arima0.0.1.H1-test.noTendEst.H1)^2)
cat("error.test.arima0.0.1.H1:", error.test.arima0.0.1.H1, "\n")
```

```
## error.test.arima0.0.1.H1: 270.4084
```

```r
plot.ts(train.noTendEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(-15, 12))
lines(ajustados.arima0.0.1.H1, col = "blue")
lines(tiempo.test, test.noTendEst.H1, col = "red")
lines(tiempo.test, pred.arima0.0.1.H1, col = "blue")
```



Conclusiones.

Vamos a comparar los valores de los errores cuadráticos acumulados de los 4 modelso propuestos:

```r
list(error.test.arima1.1.0.H1 = error.test.arima1.1.0.H1,
     error.test.arima2.2.0.H1 = error.test.arima2.2.0.H1,
     error.test.arima0.0.1.H1 = error.test.arima0.0.1.H1)
```

```
## $error.test.arima1.1.0.H1
## [1] 144.4696
##
## $error.test.arima2.2.0.H1
## [1] 6422.443
##
## $error.test.arima0.0.1.H1
```

```
## [1] 270.4084
```

Como podemos ver, el menor error lo obtenemos con el modelo pensado en base a los resultados del análisis de la serie temporal.

Por lo tanto, vamos a validar el nuevo modelo aplicando los siguientes tests estadísticos:

- Test de Box-Pierce para aleatoriedad de residuos.

```
#Tests para la seccion del modelo y su validacion
box.pierce <- Box.test(arima0.0.1.H1$residuals)
box.pierce
```

```
##
##  Box-Pierce test
##
## data:  arima0.0.1.H1$residuals
## X-squared = 6.8796, df = 1, p-value = 0.008718
```

Obtenemos un *p-value* de 0.0087184, lo cual nos indica que el test se pasa y, por lo tanto, los errores son aleatorios. Esto nos indica que nuestro modelo modela todo lo que puede modelar.

- Tests de Jarque Bera y Shapiro-Wilk para normalidad de residuos.

```
#Test de normalidad de Jarque Bera
jbt <- jarque.bera.test(arima0.0.1.H1$residuals)
jbt
```

```
##
##  Jarque Bera Test
##
## data:  arima0.0.1.H1$residuals
## X-squared = 2.1109, df = 2, p-value = 0.348
```

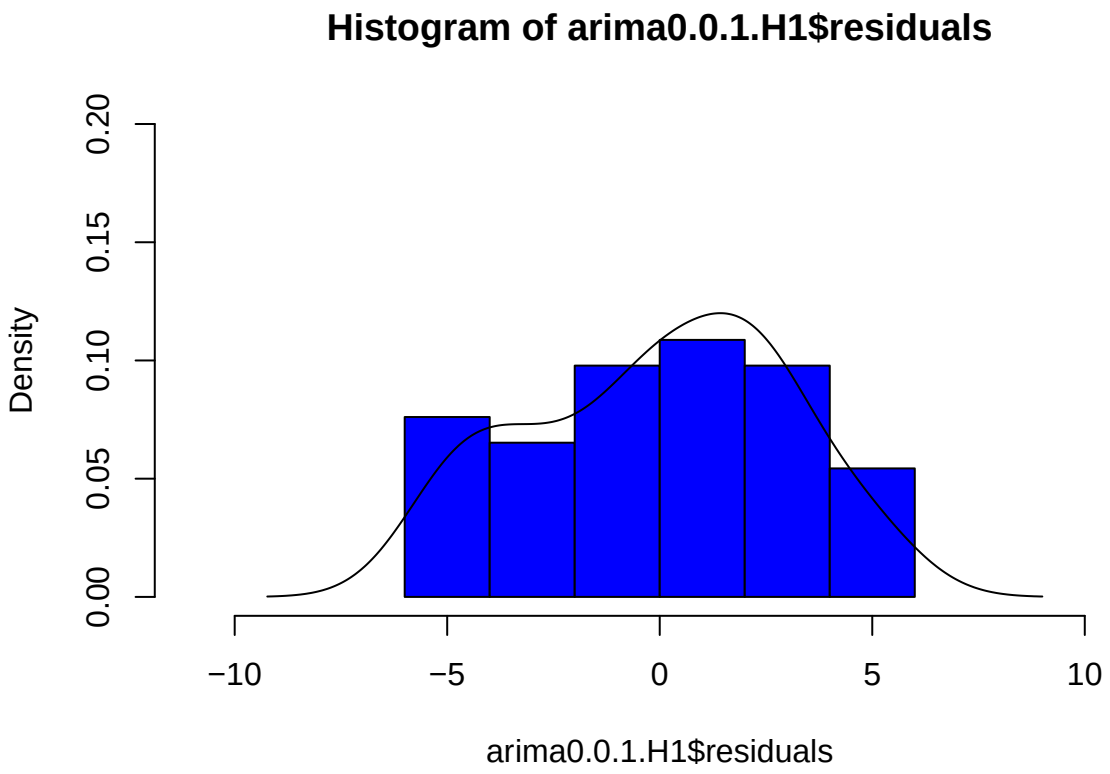
```
#Test de normalidad de Shaphiro-Wilk
saphiro <- shapiro.test(arima0.0.1.H1$residuals)
saphiro
```

```
##
##  Shapiro-Wilk normality test
##
## data:  arima0.0.1.H1$residuals
## W = 0.95868, p-value = 0.1018
```

Obtenemos un *p-value* de 0.3480428 y 0.1018106 respectivamente, por lo tanto, ambos tests pasan. Esto nos indica que los residuos son normales.

- Mostramos un histograma y la función de densidad para confirmación gráfica.

```
hist(arima0.0.1.H1$residuals, col = "blue", prob = T, ylim=c(0, 0.2), xlim=c(-11, 11))
lines(density(arima0.0.1.H1$residuals))
```

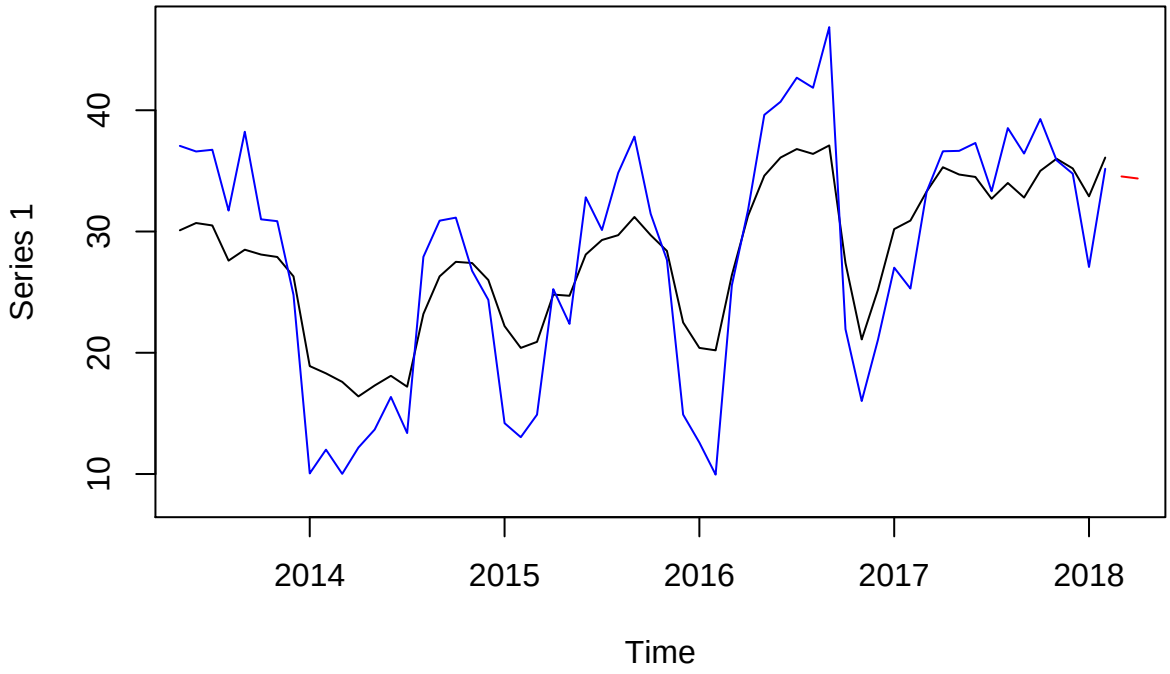


Predicción.

Una vez que hemos validado todo el modelo, volvemos a seguir los pasos iniciales, sin dividir la serie en ajuste y test, para hacer la predicción de los datos nuevos:

```
tiempo <- 1:length(serie.ts)

parametros <- lm(serie.ts ~ tiempo) #Ajustamos modelo de tendencia

TendEstimada <- parametros$coefficients[1] + tiempo * parametros$coefficients[2]
serieSinTend <- serie.ts - TendEstimada

aux <- decompose(serie.ts)$seasonal
estacionalidad <- as.numeric(aux[1:12])
aux <- rep(estacionalidad, length(serie.ts) / length(estacionalidad))
serieSinTendEst <- serieSinTend - aux
modelo <- arima(serieSinTend, order = c(0, 0, 1))
valoresAjustados <- serieSinTend + modelo$residuals
predicciones <- predict(modelo, n.ahead = 2)
# Cogemos las predicciones
valoresPredichos <- predicciones$pred

# Por ultimo, deshacemos la estacionalidad
valoresAjustados <- valoresAjustados + aux
# Cogemos los elementos 11:12 de la estacionalidad ya que son los asociados
# a los meses de marzo y abril, que son los meses que queremos predecir
valoresPredichos <- valoresPredichos + estacionalidad[11:12]
```

```
valoresAjustados <- valoresAjustados + TendEstimada
tiempoPred <- (tiempo[length(tiempo)]+(1:2))
TendEstimadaPred <- parametros$coefficients[1] + tiempoPred*parametros$coefficients[2]
valoresPredichos <- valoresPredichos + TendEstimadaPred

tiempoPred <- (tiempo[length(tiempo)]+(1:n.test))

plot.ts(serie.ts, xlim=c(2013.4, 2018.2), ylim=c(8, 47))
lines(valoresAjustados, col = "blue")
lines(valoresPredichos, col = "red")
```



Finalmente, la respuesta a la pregunta *¿Qué valores de temperatura máxima, a escala diaria, se espera para la primera semana de Marzo de 2018?* es:

```
valoresPredichos
```

```
##           Mar      Apr
## 2018 34.54090 34.37309
```

Serie diaria

Limpiamos el *environment* de trabajo:

```
rm(list = ls(all = T))
```

Se ha escogido la estación meteorológica con el identificador *5530E*, ubicada en el Aeropuerto de Granada.

```
serie <- read.csv("5530E.csv", sep = ";")
```

Como podemos ver, tenemos 1754 valores en la serie. Dado que son muchos valores, nos vamos a quedar sólo con los valores de este año, es decir, desde enero (he descartado los tres primeros días de Enero para así tener 56 elemntos, múltiplo de 7, ya que, como comento más adelante, voy a suponer una frecuencia y estacionalidad de 7):

```
serie <- serie[1699:nrow(serie),]
```

A continuación, dado que la información que nos interesa es la temperatura máxima, el estudio lo vamos a hacer sobre la columna *Tmax* de nuestro conjunto de datos:

```
serieTmax <- serie$Tmax
```

Finalmente, convertimos los datos a una serie temporal. La frecuencia se ha puesto a 7 para representar una frecuencia semanal.

```
serie.ts <- ts(serieTmax, frequency = 7)
```

Veamos si hay valores perdidos en la serie:

```
sum(is.na(serieTmax))
```

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

Como podes ver, los hay, así que vamos a aplicar preprocesamiento.

Preprocesamiento

Para la imputación de valores perdidos se ha usado el paquete *imputeTs* (https://cran.r-project.org/web/packages/imputeTS/index.html):

```
serie <- imputeTS::na.ma(serie.ts)
serie.ts <- ts(serie, frequency = 7)
```

Veamos si sigue habiendo valores perdidos:

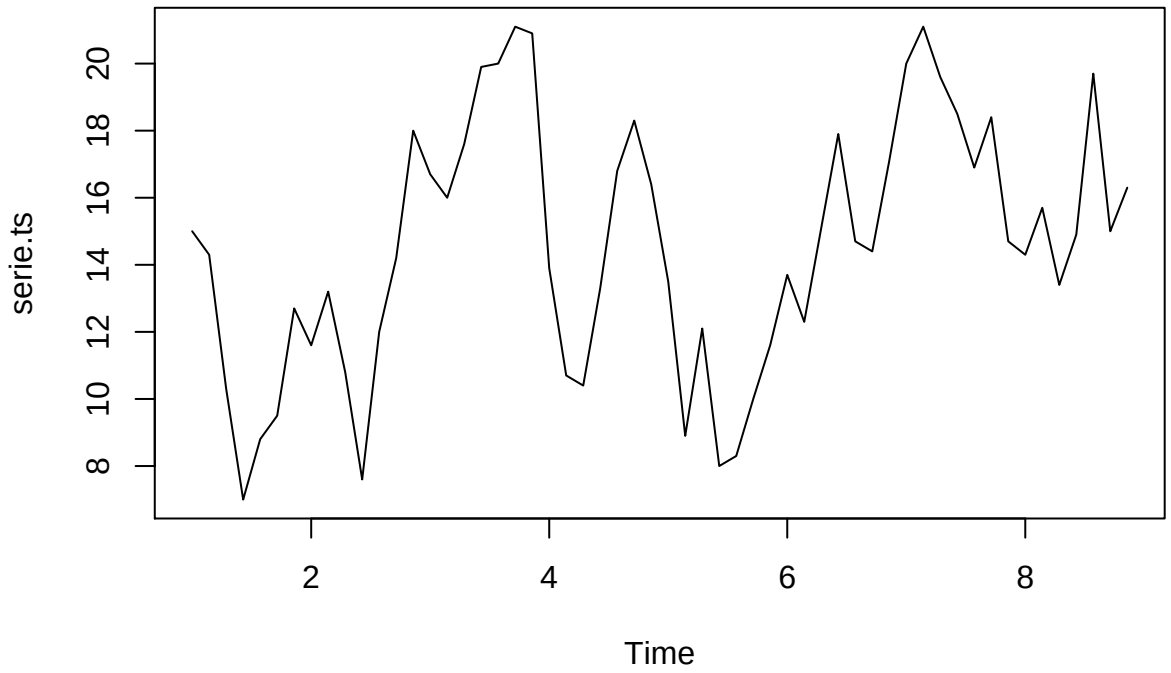
```
sum(is.na(serie))
```

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

Como podes ver, ya no hay valores perdidos en la serie.

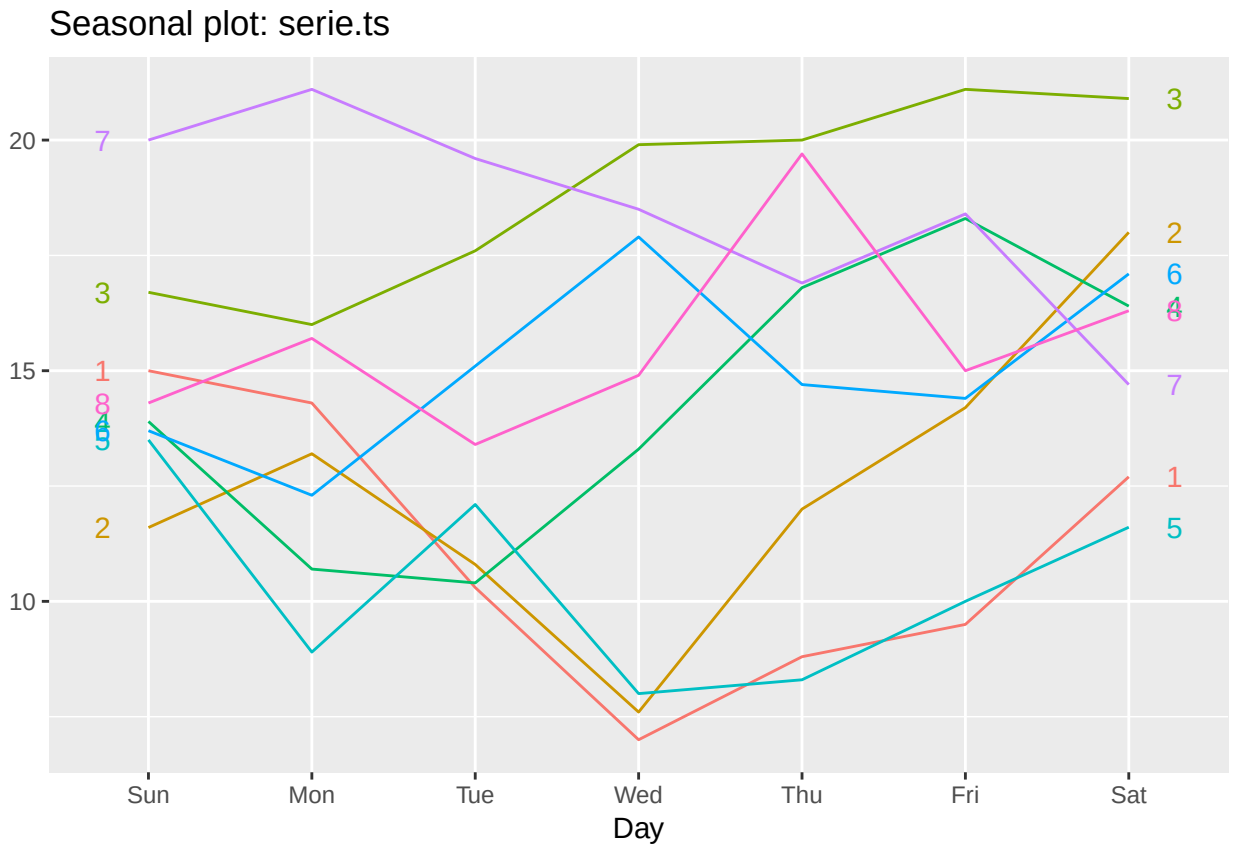
Análisis inicial de la serie.

```
plot.ts(serie.ts)
```



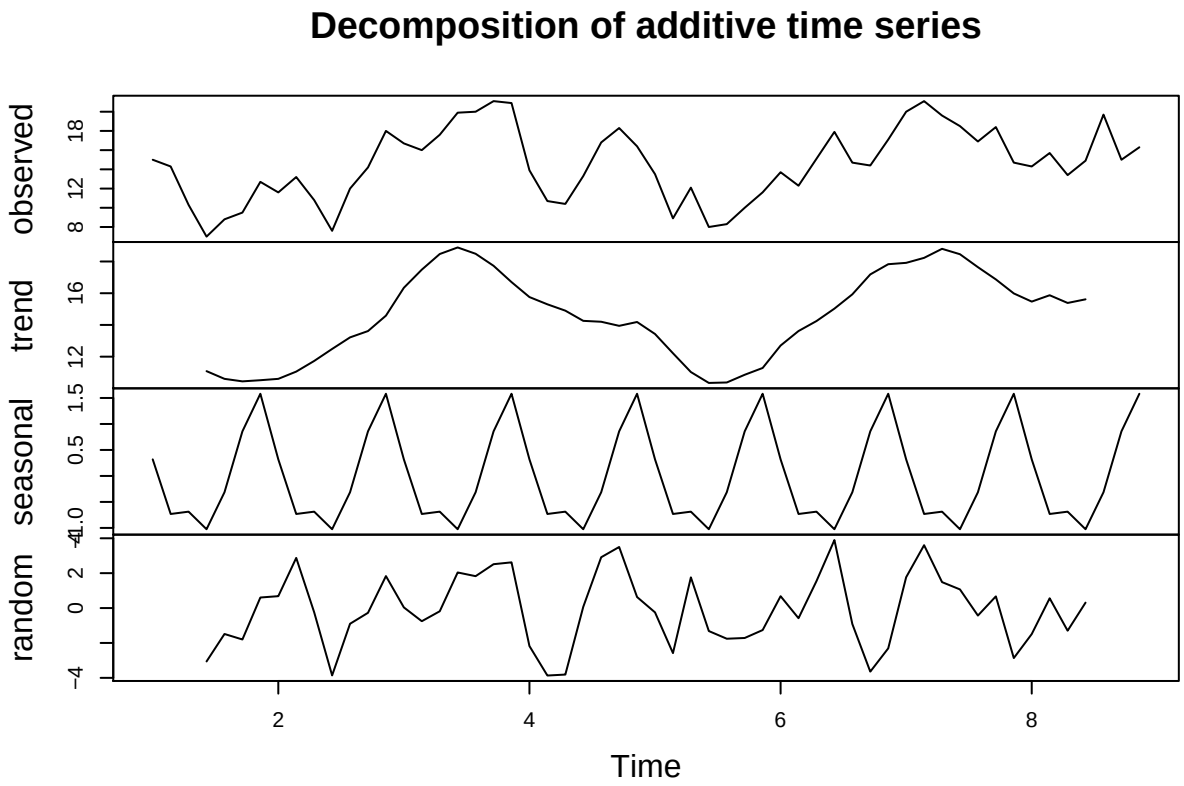
Como podemos ver, no hay ninguna tendencia en nuestra serie, ya que la temperatura no tiene a subir o a bajar a lo largo del tiempo, pero si se puede apreciar cierta estacionalidad con la que tendremos que lidiar. Al igual que en el trabajo guiado, podemos usar la función *ggseasonplot* del parquete *forecast* para ver más información acerca de la tendencia de la serie temporal (idea sacada de la página web https://otexts.com/fpp2/graphics.html):

```
forecast::ggseasonplot(serie.ts, year.labels = T, year.labels.left = T)
```



Analizando este gráfico podemos confirmar, una ves más, que no hay tendencia en la serie, ya que no se ve ni un crecimiento ni un decrecimiento de la temperatura según avanza el tiempo. Veamos la descomposición de la serie:

```
plot(decompose(serie.ts))
```

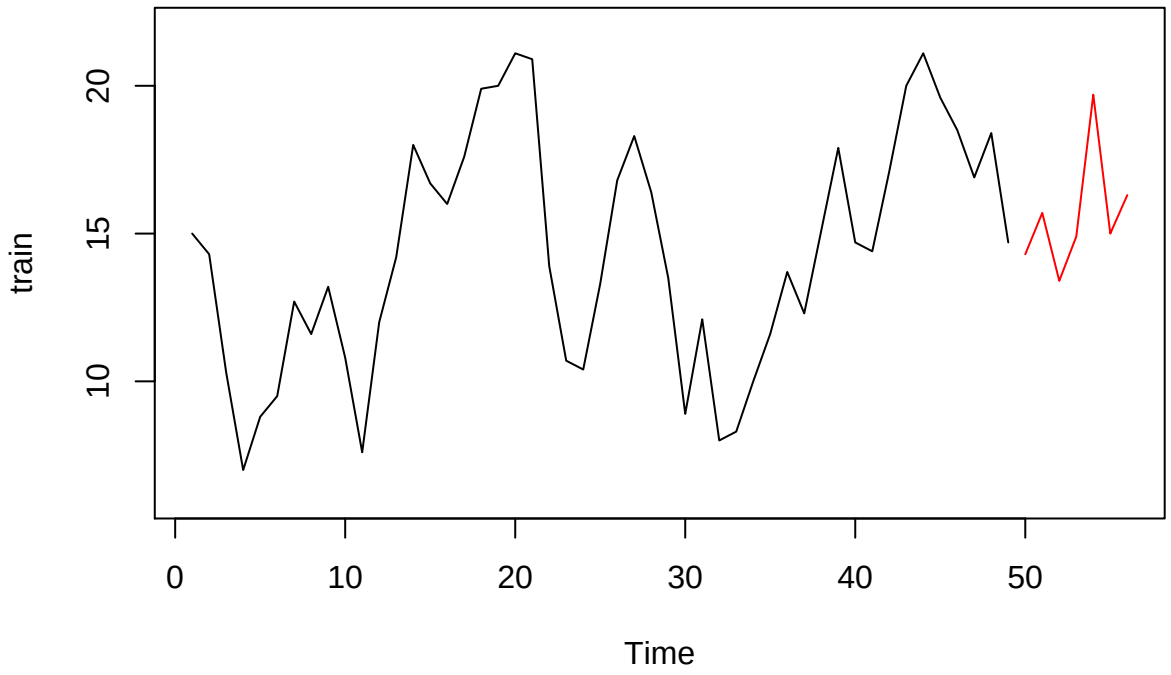


Una vez más vemos que no hay tendencia (tercera gráfica) en los datos. También se puede observar que la estacionalidad no tiene tendencia (segunda gráfica) y, por lo tanto, no se ha aplicado ninguna transformación de los datos como se hizo en el trabajo guiado. Finalmente, podemos observar un fuerte patrón estacional (segunda gráfica).

Particionamiento en train y test.

Se ha seleccionado como *test* la última semana de datos disponible y el resto como conjunto de *train*.

```
n.test <- 7
train <- serie[1:(length(serie) - n.test)]
tiempo.train <- 1:length(train)
test <- serie[(length(serie) - n.test + 1):length(serie)]
tiempo.test <- (tiempo.train[length(tiempo.train)] + 1):
  (tiempo.train[length(tiempo.train)] + n.test)

plot.ts(train, xlim = c(1, tiempo.test[length(tiempo.test)]), ylim=c(6, 22))
lines(tiempo.test, test, col = "red")
```



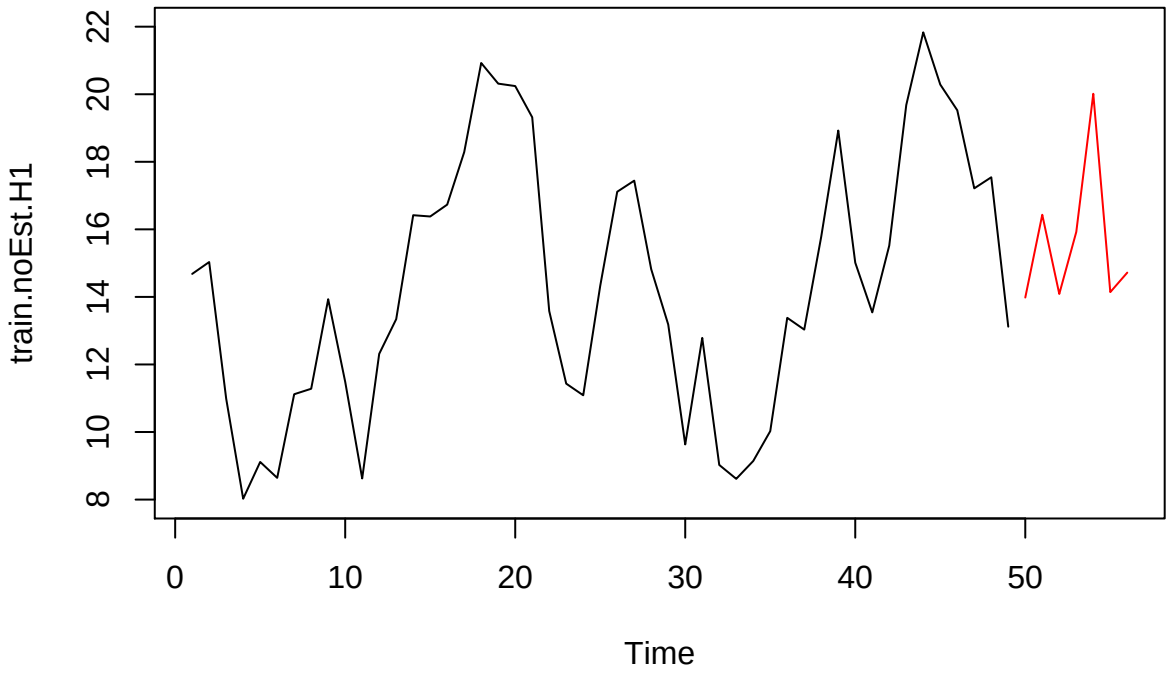
Análisis de Tendencia.

Tal y como se ha comentado en la sección de *Análisis inicial de la serie*, la serie no presenta tendencia así que no tenemos que hacer ningún análisis de la misma.

Análisis de Estacionalidad.

El siguiente paso es eliminar la estacionalidad. Para ello, vamos a usar un modelo de diferenciación. Asumimos una estacionalidad semanal. Para eliminar la estacionalidad, podemos hacer uso de las salidas de la función "decompose":

```
# Asumimos periodo de estacionalidad k = 7
k <- 7
estacionalidad <- decompose(serie.ts)$seasonal[1:k]

# Eliminamos estacionalidad para el modelo
aux <- rep(estacionalidad, length(train) / length(estacionalidad))

train.noEst.H1 <- train - aux
test.noEst.H1 <- test - estacionalidad

plot.ts(train.noEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(8, 22))
lines(tiempo.test, test.noEst.H1, col = "red")
```



Análisis de Estacionariedad.

Con la serie sin tendencia ni estacionalidad, debemos comprobar si es estacionaria antes de hipotetizar modelos de predicción. Usamos el test de Dickey-Fuller aumentado.

```
# Comprobamos el test de Dickye-Fuller aumentado para la estacionaridad
adf <- adf.test(train.noEst.H1)
adf
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  train.noEst.H1
## Dickey-Fuller = -2.4182, Lag order = 3, p-value = 0.4068
## alternative hypothesis: stationary
```

Al no superarse el test (obtenemos un *p-value* de 0.4068327, el cual es mayor que 0.05) podemos decir que la serie no es estacionaria. Para solucionar esto, diferenciamos la serie:

```
# Diferenciamos la serie
train.noEstDiff.H1 <- diff(train.noEst.H1)
test.noEstDiff.H1 <- diff(test.noEst.H1)
```

Una vez diferenciada, volvemos a aplicar el test:

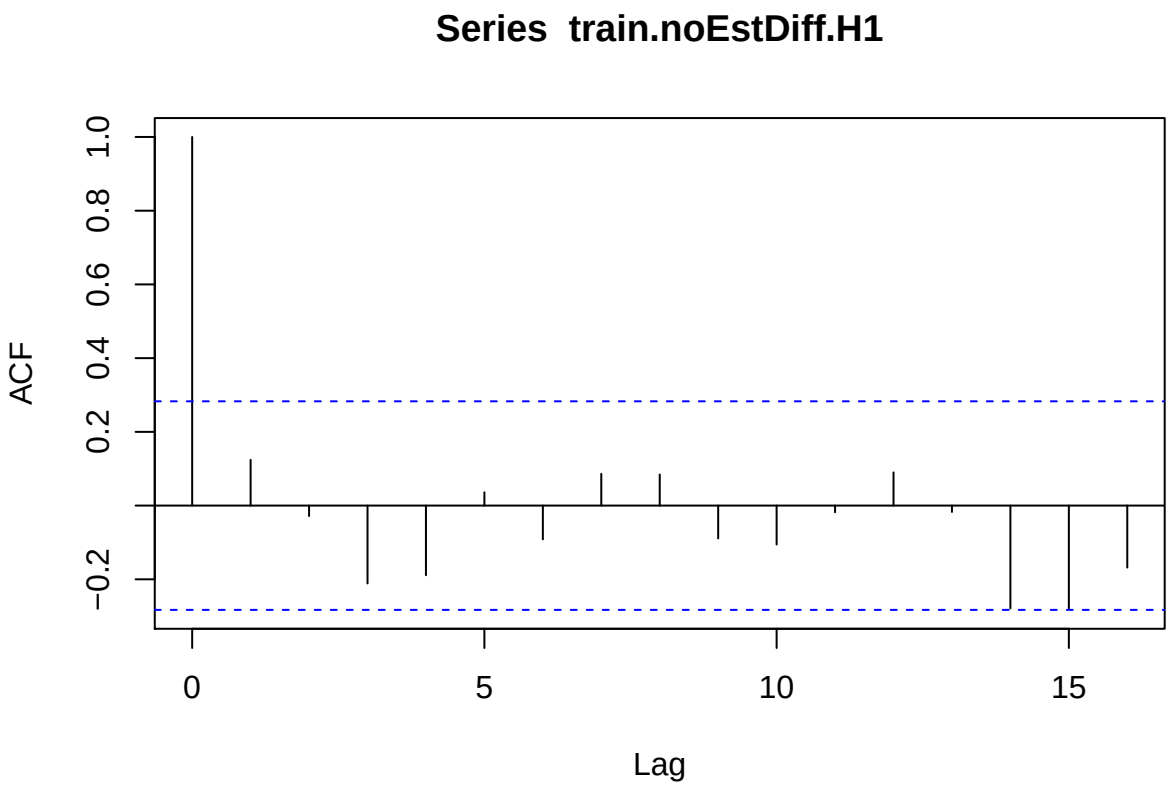
```
# Volvemos a aplicar el test
adf <- adf.test(train.noEstDiff.H1)
adf
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  train.noEstDiff.H1
## Dickey-Fuller = -4.0711, Lag order = 3, p-value = 0.01444
## alternative hypothesis: stationary
```

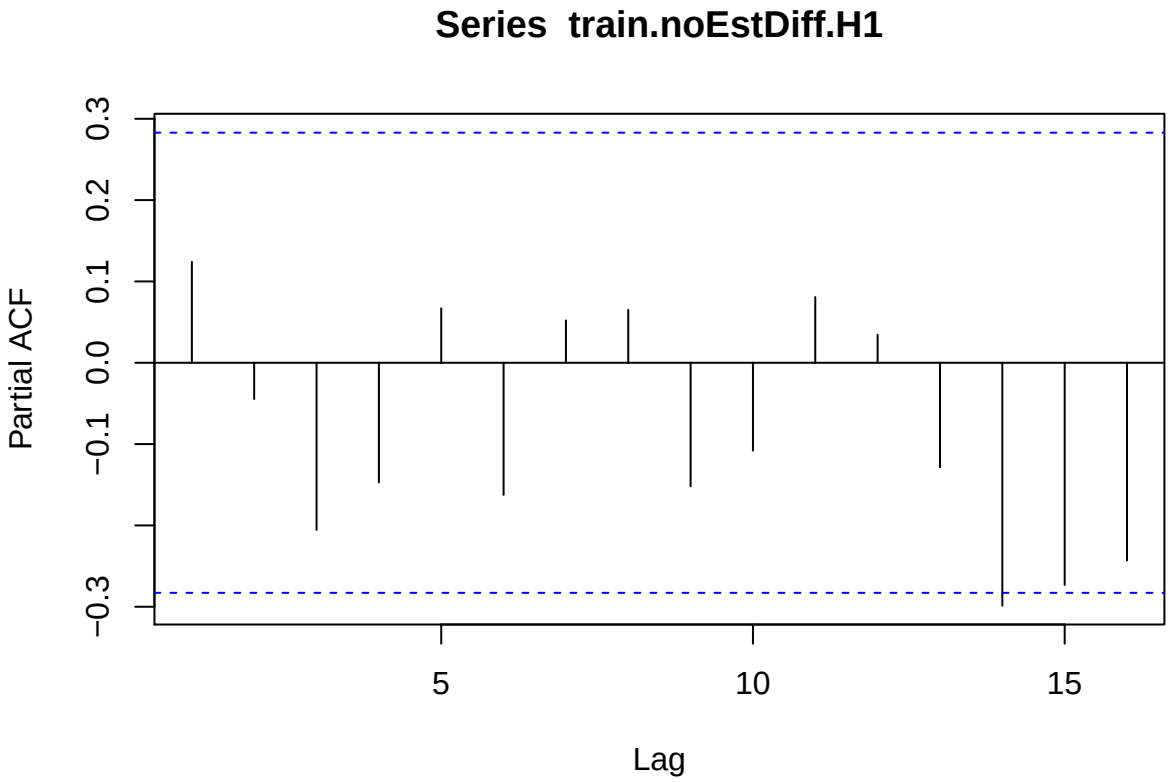
Podemos ver que, tras aplicar diferenciación, la serie que tenemos si es estacionaria (ya que obtiene un *p-value* de 0.0144431, el cual es menor que 0.05).

Finalmente, podemos comprobar que la serie es estacionaria vamos a visualizar el gráfico ACF y el gráfico PACF:

```
acf(train.noEstDiff.H1)
```




```
pacf(train.noEstDiff.H1)
```



Selección del modelo de predicción.

Viendo los gráficos ACF y PACF podemos de decir son los gráficos típicos de un modelo autoregresivo. Para ver de que orden, sólo debemos coger el índice del último *lag* que pasa los umbrales en el gráfico PACF. En mi caso, nos encontramos ante un modelo autoregresivo de orden 14, es decir, un *AR(14)*. Cómo hemos diferenciado una vez, podríamos incluir la diferencia dentro del modelo ajustando un *ARIMA(14, 1, 0)* sobre la serie sin tendencia y sin estacionalidad.

```
# Ajustamos el modelo
arima14.1.0.H1 <- arima(train.noEst.H1, order = c(14, 1, 0))
```

Validación del modelo de predicción.

Para validad el modelo usado para le predicción voy calcular las los valores de la serie temporal para el conjunto de test y voy a calcular el error cuadrático acumulado del ajuste, tanto en el proceso de entrenamiento como en el proceso de test:

```
ajustados.arima14.1.0.H1 <- train.noEst.H1 + arima14.1.0.H1$residuals

# Calculamos las predicciones
pred.arima14.1.0.H1 <- predict(arima14.1.0.H1, n.ahead = n.test)$pred

# Calculamos el error cuadrático acumulado del ajuste, en ajuste y en test
error.train.arima14.1.0.H1 <- sum((arima14.1.0.H1$residuals)^2)
cat("error.train.arima14.1.0.H1:", error.train.arima14.1.0.H1, "\n")
```

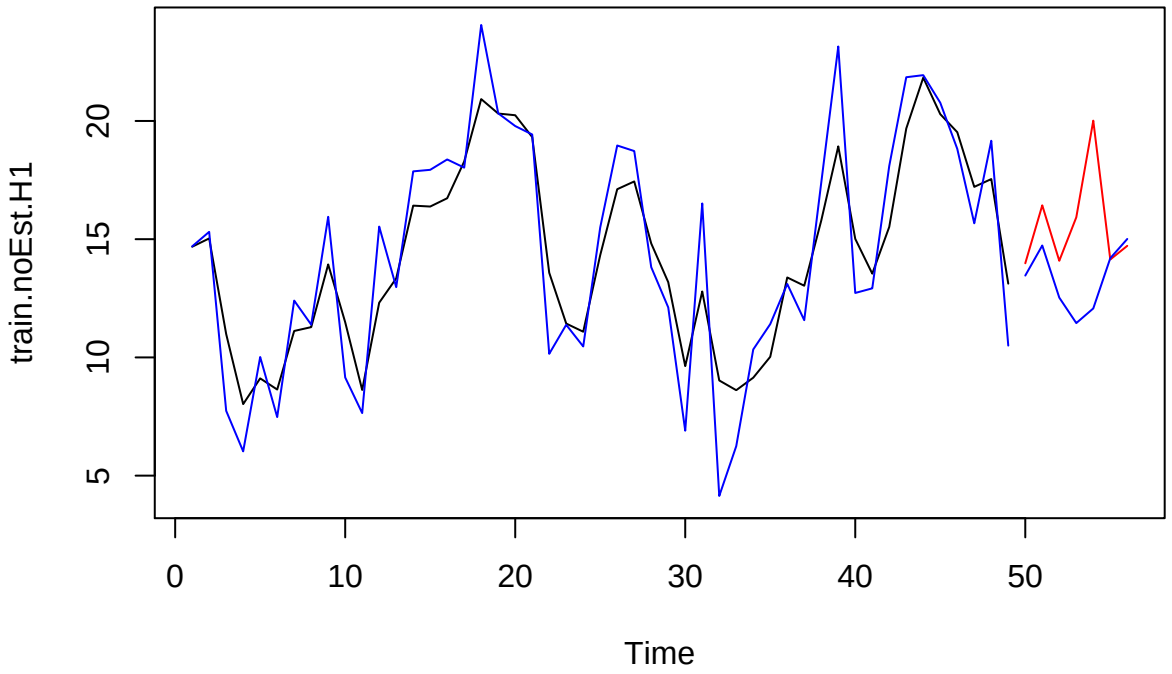
```
## error.train.arima14.1.0.H1: 183.8117
```

```
error.test.arima14.1.0.H1 <- sum((pred.arima14.1.0.H1-test.noEst.H1)^2)
cat("error.test.arima14.1.0.H1:", error.test.arima14.1.0.H1, "\n")
```

```
## error.test.arima14.1.0.H1: 88.75674
```

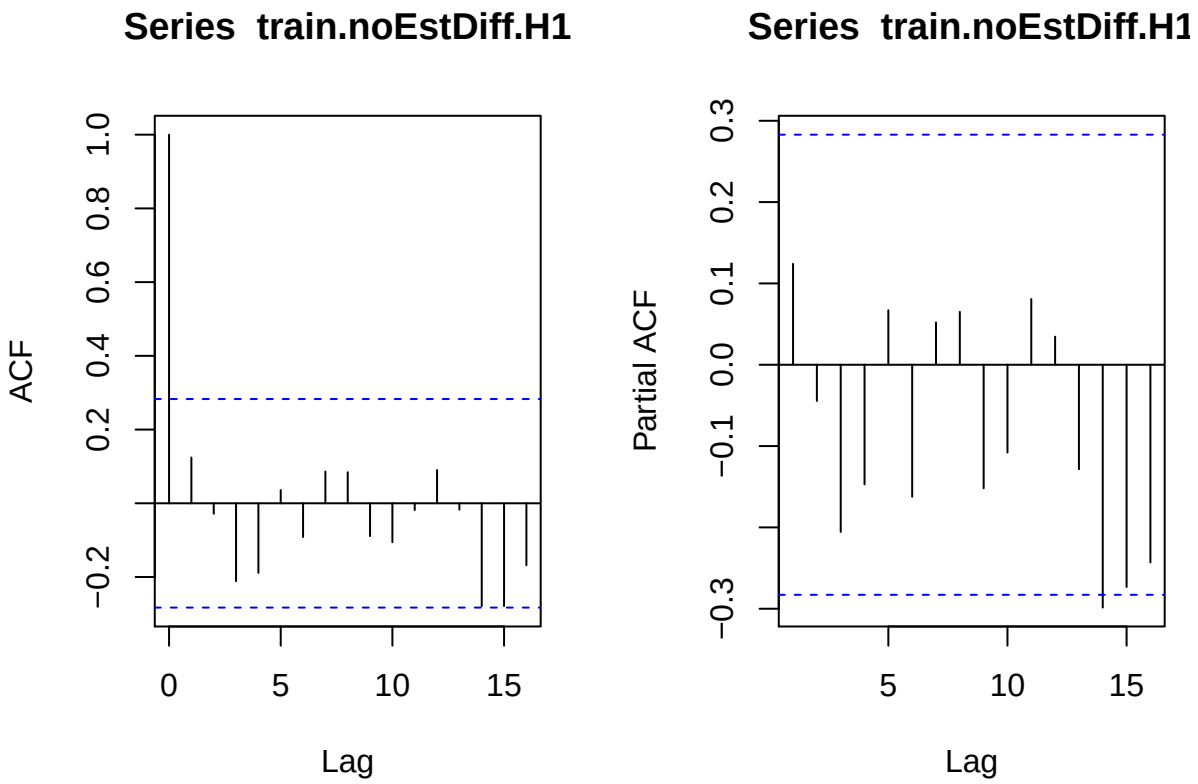
También voy a mostrar las gráficas del ajuste y predicción en test:

```
plot.ts(train.noEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(4, 24))
lines(ajustados.arima14.1.0.H1, col = "blue")
lines(tiempo.test, test.noEst.H1, col = "red")
lines(tiempo.test, pred.arima14.1.0.H1, col = "blue")
```



Si nos fijamos en la como nuestro modelo predice el conjunto de test, podríamos decir que no lo hace del todo bien. Si observamos de nuevo el gráfico ACF y PACF de la serie que acabamos de modelar con nuestro modelo ARIMA:

```
par(mfrow=c(1,2))
acf(train.noEstDiff.H1)
pacf(train.noEstDiff.H1)
```




```
par(mfrow=c(1,2))
```

En el gráfico ACF podemos observar que el dato importante para predicir un dato es el propio dato. En el gráfico PACF podemos ver que sólo el *lag* 14 supera el umbral, esto quiere decir que para predecir un dato, necesitamos ese dato y los catorce anteriores. Aún así, supera el umbral por muy poco. Estos dos gráficos nos indican que estamos ante una serie que es ruido blanco.

Por lo tanto, vamos a validar el modelo aplicando los siguientes tests estadísticos:

- Test de Box-Pierce para aleatoriedad de residuos.

```
#Tests para la seccion del modelo y su validacion
box.pierce <- Box.test(arima14.1.0.H1$residuals)
box.pierce
```

```
##
##  Box-Pierce test
##
## data:  arima14.1.0.H1$residuals
## X-squared = 0.55911, df = 1, p-value = 0.4546
```

Obtenemos un *p-value* de 0.454618, lo cual nos indica que el test se pasa y, por lo tanto, los errores son aleatorios. Esto nos indica que nuestro modelo modela todo lo que puede modelar.

- Tests de Jarque Bera y Shapiro-Wilk para normalidad de residuos.

```
#Test de normalidad de Jarque Bera
jbt <- jarque.bera.test(arima14.1.0.H1$residuals)
jbt
```

```
##
##  Jarque Bera Test
##
```

```
## data:  arima14.1.0.H1$residuals
## X-squared = 0.29798, df = 2, p-value = 0.8616
```

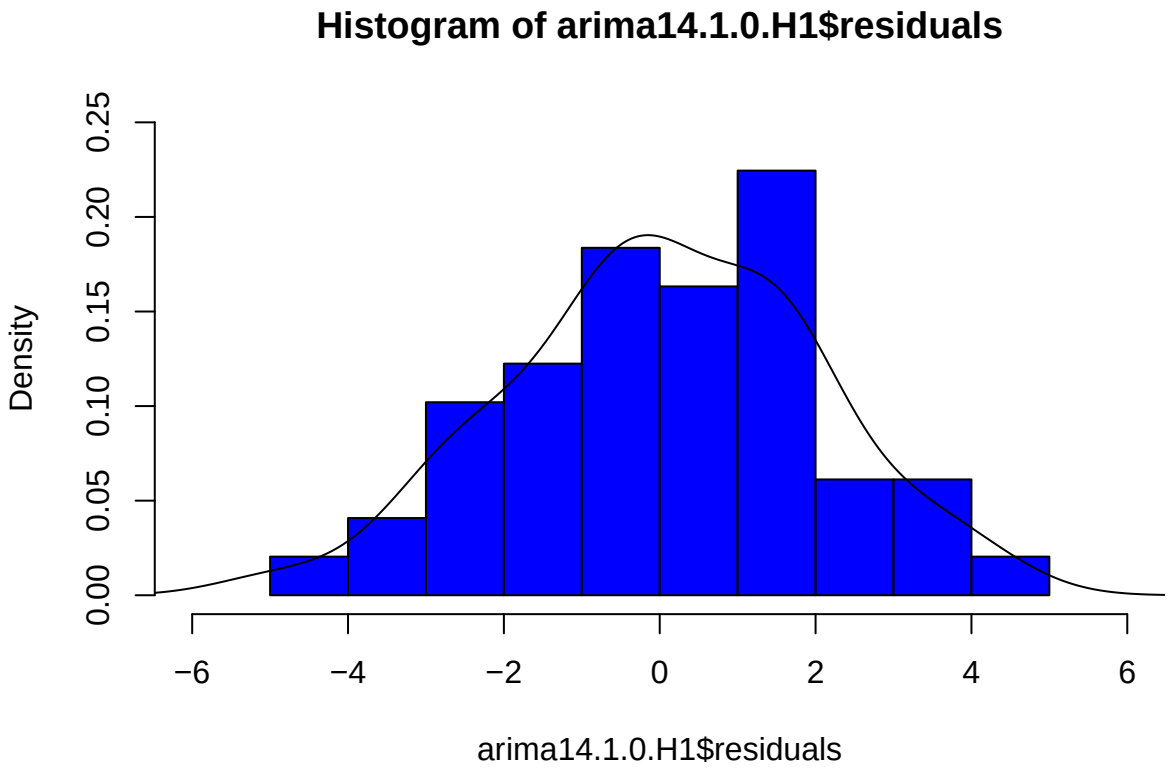
```
#Test de normalidad de Shaphiro-Wilk
saphiro <- shapiro.test(arima14.1.0.H1$residuals)
saphiro
```

```
##
##  Shapiro-Wilk normality test
##
## data:  arima14.1.0.H1$residuals
## W = 0.99046, p-value = 0.9592
```

Obtenemos un *p-value* de 0.8615793 y 0.9592257 respectivamente, por lo tanto, ambos tests pasan. Esto nos indica que los residuos son normales.

- Mostramos un histograma y la función de densidad para confirmación gráfica.

```
hist(arima14.1.0.H1$residuals, col = "blue", prob = T, ylim=c(0, 0.25), xlim=c(-6, 6))
lines(density(arima14.1.0.H1$residuals))
```



Comparación con otros modelos.

Al igual que hemos probado un modelo *ARIMA(14, 1, 0)*, voy a probar distintos modelos. Para comparar los distintos modelos se va a usar el valor del error cuadrático acumulado. Otra opción de métrica estudiada en la asignatura para comprar los modelos es el *AIC*. El problema que tiene el *AIC* es que es una métrica que se obtiene sólo en base a como funciona el modelo para los datos empleados en el proceso de entrenamiento. Sin embargo, el error cuadrático acumulado permite saber como se comporta el modelo para un conjunto de test el cual nunca ha visto.

ARIMA(1, 1, 0).

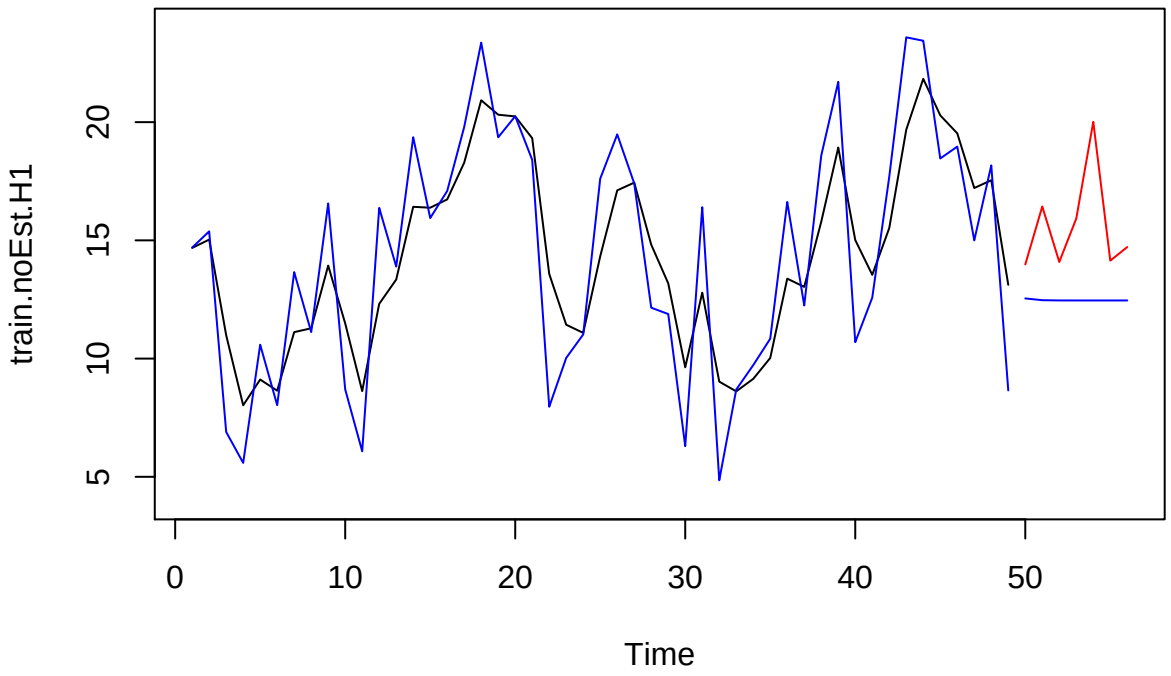
```
# Ajustamos el modelo
arima1.1.0.H1 <- arima(train.noEst.H1, order = c(1, 1, 0))

ajustados.arima1.1.0.H1 <- train.noEst.H1 + arima1.1.0.H1$residuals

# Calculamos las predicciones
pred.arima1.1.0.H1 <- predict(arima1.1.0.H1, n.ahead = n.test)$pred

# Calculamos el error cuadrático acumulado del ajuste, en ajuste y en test
error.train.arima1.1.0.H1 <- sum((arima1.1.0.H1$residuals)^2)
cat("error.train.arima1.1.0.H1:", error.train.arima1.1.0.H1, "\n")
```

```
## error.train.arima1.1.0.H1: 287.7248
```

```
error.test.arima1.1.0.H1 <- sum((pred.arima1.1.0.H1-test.noEst.H1)^2)
cat("error.test.arima1.1.0.H1:", error.test.arima1.1.0.H1, "\n")
```

```
## error.test.arima1.1.0.H1: 97.45225
```

```
plot.ts(train.noEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(4, 24))
lines(ajustados.arima1.1.0.H1, col = "blue")
lines(tiempo.test, test.noEst.H1, col = "red")
lines(tiempo.test, pred.arima1.1.0.H1, col = "blue")
```



ARIMA(2, 1, 0).

```
# Ajustamos el modelo
arima2.1.0.H1 <- arima(train.noEst.H1, order = c(2, 1, 0))

ajustados.arima2.1.0.H1 <- train.noEst.H1 + arima2.1.0.H1$residuals

# Calculamos las predicciones
```

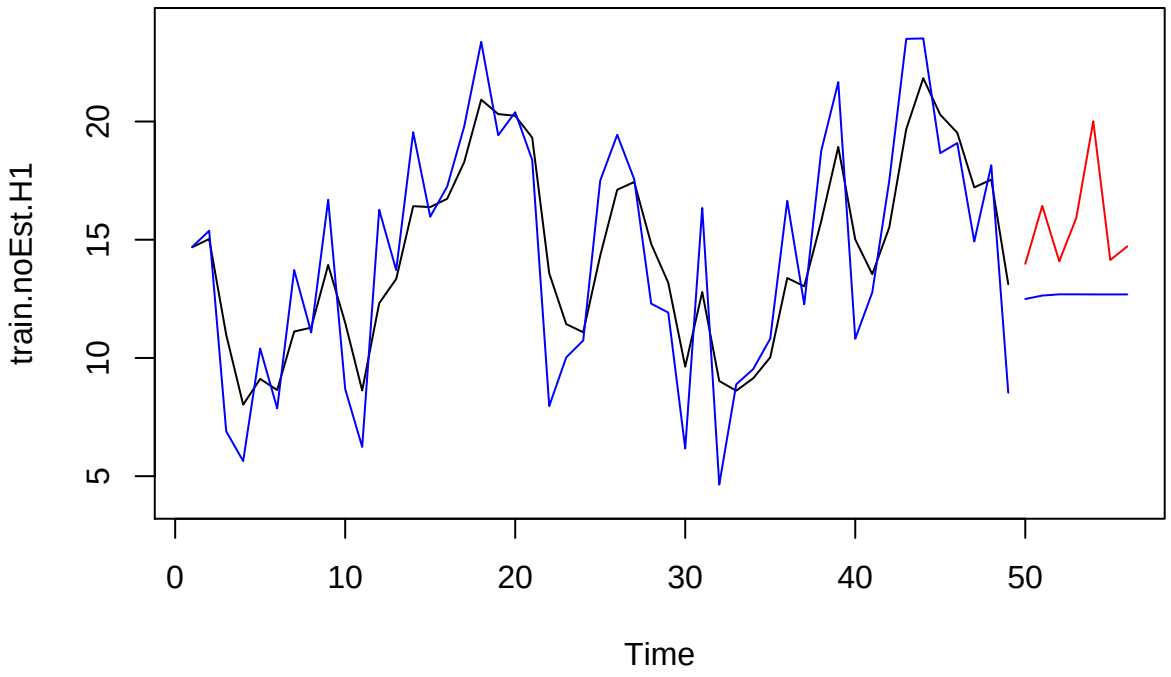
```r
pred.arima2.1.0.H1 <- predict(arima2.1.0.H1, n.ahead = n.test)$pred

# Calculamos el error cuadrático acumulado del ajuste, en ajuste y en test
error.train.arima2.1.0.H1 <- sum((arima2.1.0.H1$residuals)^2)
cat("error.train.arima2.1.0.H1:", error.train.arima2.1.0.H1, "\n")
```

```
## error.train.arima2.1.0.H1: 286.9759
```

```r
error.test.arima2.1.0.H1 <- sum((pred.arima2.1.0.H1-test.noEst.H1)^2)
cat("error.test.arima2.1.0.H1:", error.test.arima2.1.0.H1, "\n")
```

```
## error.test.arima2.1.0.H1: 88.93241
```

```r
plot.ts(train.noEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(4, 24))
lines(ajustados.arima2.1.0.H1, col = "blue")
lines(tiempo.test, test.noEst.H1, col = "red")
lines(tiempo.test, pred.arima2.1.0.H1, col = "blue")
```



ARIMA(2, 2, 0).

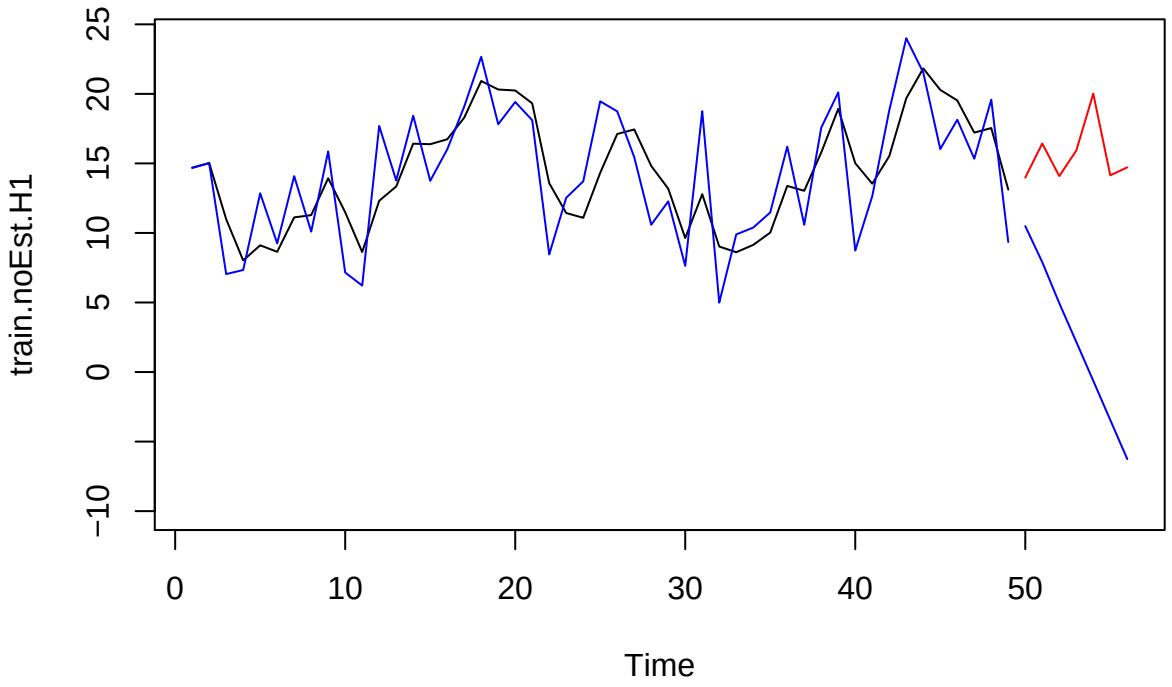
```r
# Ajustamos el modelo
arima2.2.0.H1 <- arima(train.noEst.H1, order = c(2, 2, 0))

ajustados.arima2.2.0.H1 <- train.noEst.H1 + arima2.2.0.H1$residuals

# Calculamos las predicciones
pred.arima2.2.0.H1 <- predict(arima2.2.0.H1, n.ahead = n.test)$pred

# Calculamos el error cuadrático acumulado del ajuste, en ajuste y en test
error.train.arima2.2.0.H1 <- sum((arima2.2.0.H1$residuals)^2)
cat("error.train.arima2.2.0.H1:", error.train.arima2.2.0.H1, "\n")
```

```
## error.train.arima2.2.0.H1: 398.831
```

```
error.test.arima2.2.0.H1 <- sum((pred.arima2.2.0.H1-test.noEst.H1)^2)
cat("error.test.arima2.2.0.H1:", error.test.arima2.2.0.H1, "\n")
```

```
## error.test.arima2.2.0.H1: 1533.568
```

```
plot.ts(train.noEst.H1, xlim=c(1, tiempo.test[length(tiempo.test)]), ylim=c(-10, 24))
lines(ajustados.arima2.2.0.H1, col = "blue")
lines(tiempo.test, test.noEst.H1, col = "red")
lines(tiempo.test, pred.arima2.2.0.H1, col = "blue")
```



Conclusiones.

Vamos a comparar los valores de los errores cuadráticos acumulados de los 4 modelso propuestos:

```
list(error.test.arima14.1.0.H1 = error.test.arima14.1.0.H1,
     error.test.arima2.1.0.H1 = error.test.arima2.1.0.H1,
     error.test.arima1.1.0.H1 = error.test.arima1.1.0.H1,
     error.test.arima2.2.0.H1 = error.test.arima2.2.0.H1)
```

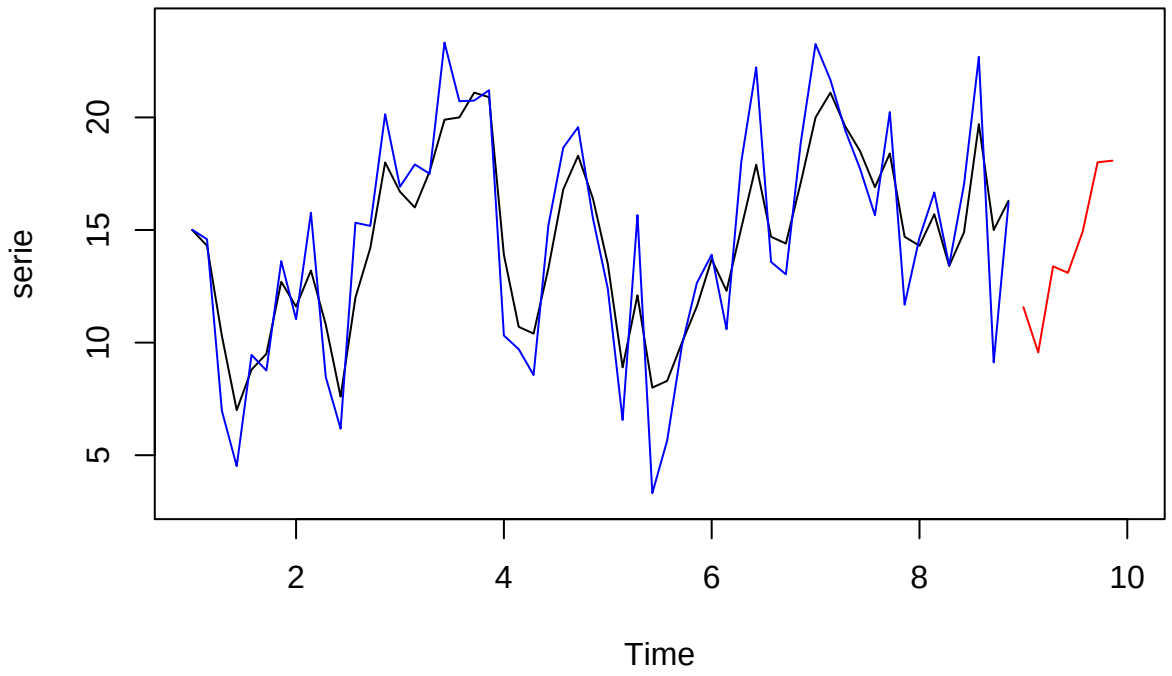
```
## $error.test.arima14.1.0.H1
## [1] 88.75674
##
## $error.test.arima2.1.0.H1
## [1] 88.93241
##
## $error.test.arima1.1.0.H1
## [1] 97.45225
##
## $error.test.arima2.2.0.H1
## [1] 1533.568
```

Como podemos ver, el menor error lo obtenemos con el modelo pensado en base a los resultados del análisis de la serie temporal. Si nos fijamos en los gráficos de los ajustes de los distintos modelos, también podemos ver que visualmente el mejor es el primer modelo propuesto.

Predicción.

Una vez que hemos validado todo el modelo, volvemos a seguir los pasos iniciales, sin dividir la serie en ajuste y test, para hacer la predicción de los datos nuevos:

```
tiempo <- 1:length(serie.ts)
aux <- ts(serie.ts, frequency = 7)
aux <- decompose(aux)$seasonal
estacionalidad <- as.numeric(aux[1:7])
aux <- rep(estacionalidad, length(serie.ts) / length(estacionalidad))
serieSinEst <- serie.ts - aux
modelo <- arima(serieSinEst, order = c(14, 1, 0))
valoresAjustados <- serieSinEst + modelo$residuals
predicciones <- predict(modelo, n.ahead = n.test)
# Cogemos las predicciones
valoresPredichos <- predicciones$pred

# Por ultimo, deshacemos lastacionalidad
valoresAjustados <- valoresAjustados + aux
valoresPredichos <- valoresPredichos + estacionalidad

tiempoPred <- (tiempo[length(tiempo)]+(1:n.test))

plot.ts(serie, xlim=c(1, 10), ylim=c(3, 24))
lines(valoresAjustados, col = "blue")
lines(valoresPredichos, col = "red")
```



Finalmente, la respuesta a la pregunta *¿Qué valores de temperatura máxima, a escala diaria, se espera para la primera semana de Marzo de 2018?* es:

```
valoresPredichos
```

```
## Time Series:
## Start = c(9, 1)
```

```
## End = c(9, 7)
## Frequency = 7
## [1] 11.573027  9.568084 13.389246 13.097218 14.937192 18.006447 18.080287
```