Análisis y predicción de Series Temporales

Alejandro Sierra Fernández

En el siguiente documento procedemos a estudiar la serie temporal que genera el Índice de Comercio al por Menor (ICM), que tiene como objetivo conocer la evolución de las ventas y el empleo en el sector del comercio minorista en España. La información se recoge de una muestra de 12.500 empresas ubicadas en todo el territorio nacional y en nuestros datos disponemos de un valor por cada mes desde enero de año 2000.

Comenzamos importando el archivo Excel a través de la función *read_excel()*, convirtiéndolo en una variable de tipo dataframe y analizando cuáles son las características principales del mismo. Cabe decir que la ruta que se emplea para cargar el archivo es una ruta relativa y no se especificarán las librerías de las que proceden cada una de las funciones que vamos a emplear en nuestro código. De esta forma:

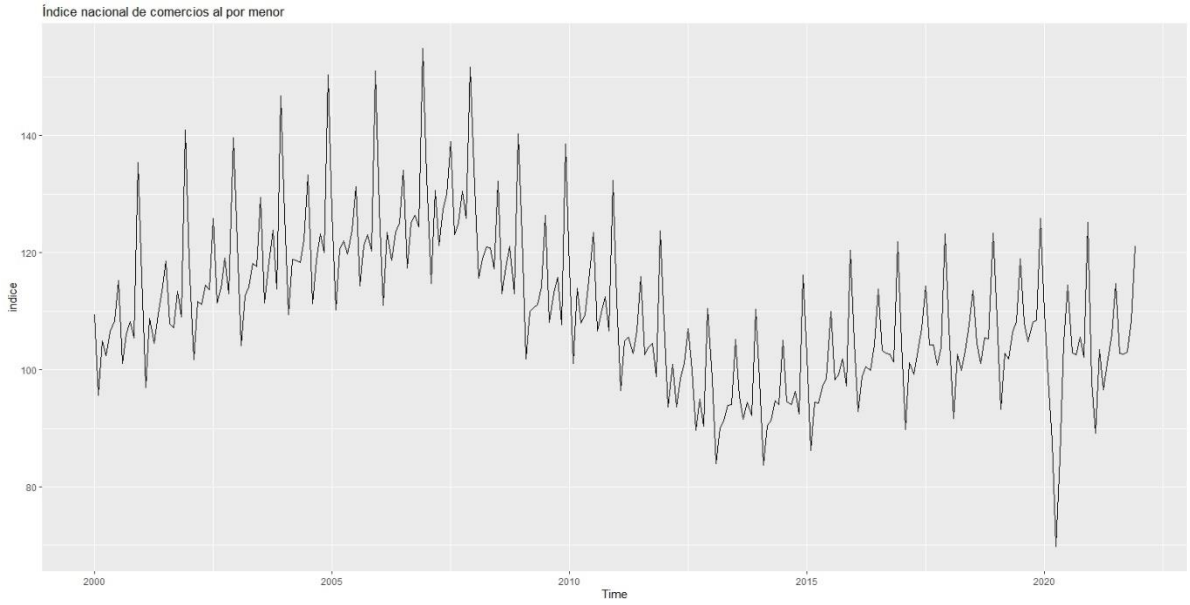
```
rm(list=ls()) #Limpiamos el entorno de trabajo

datos <-
read_excel("C:/Users/aleja/OneDrive/Escritorio/Tarea/Indice_Temp.xls")

datos <- as.data.frame(datos)

colnames(datos)

summary(datos)

str(datos)
```

Fecha	Índice
Lengths:264	Min: 69.81
Class: character	1st Qu: 101.90
Mode: character	Median: 108.69
	Mean: 110.47
	3rd Qu: 118.85
	Max: 154.85

Vemos que tenemos un dataframe con 264 instancias diferentes, y formado únicamente por dos columnas. Una reservada para la fecha y que es de tipo *"character"* y otra con el valor del índice y de tipo numérica.

Para empezar a tratar a los datos como una serie temporal y graficarla, creamos un objeto *timeseries* con la función *ts* y empleamos *autoplot* sobre este objeto:

```
indice <- ts(datos[,-1], start=c(2000,1), frequency = 12)
autoplot(indice) + ggtitle("Índice nacional de comercios al por menor")
+ xlab("mes") + ylab("índice")
```

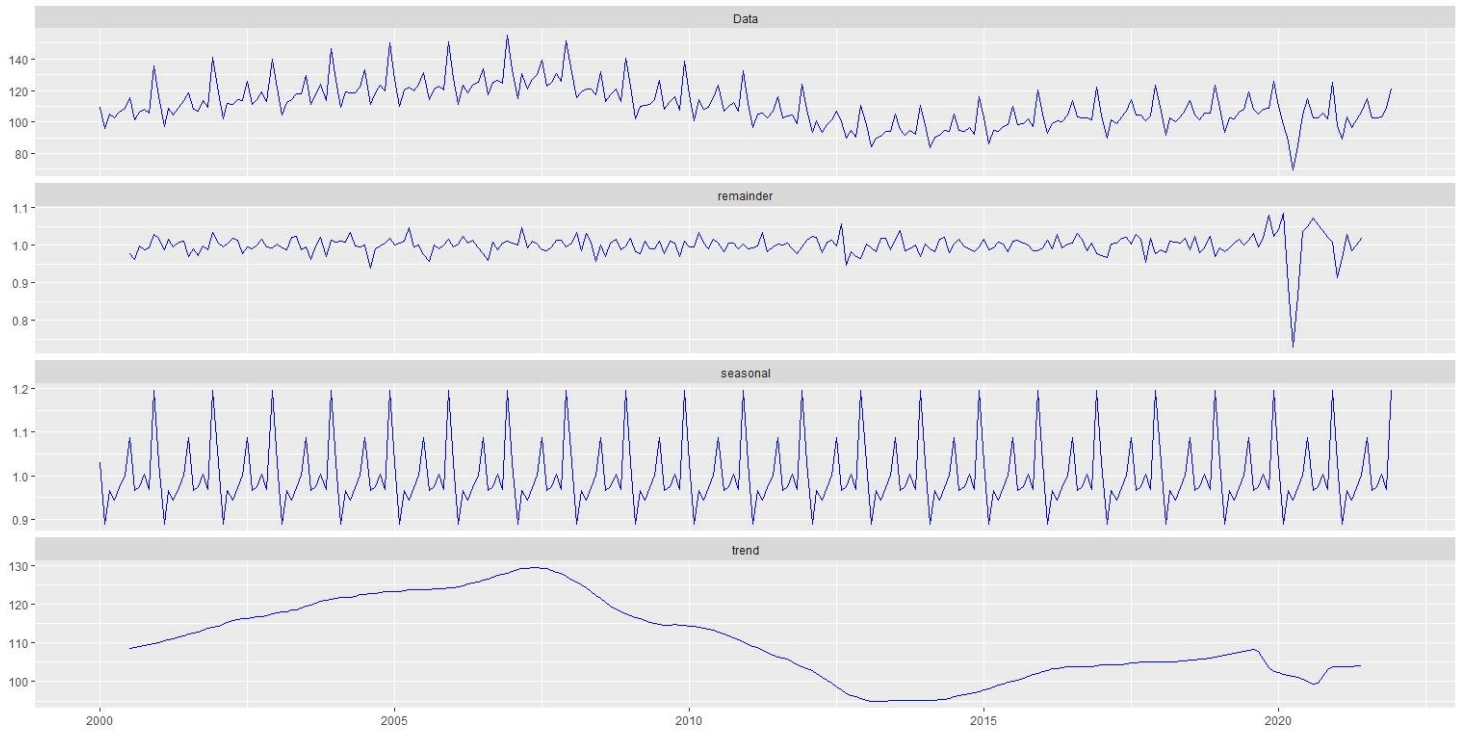


A continuación, creamos la secuencia de fechas observadas en nuestros datos para poder después crear la serie temporal con la librería *zoo*, lo que nos ofrecerá una manejabilidad de la misma que en ocasiones puede ser de mucha utilidad.

```
 dt = seq(from=as.Date("2000-01-01"), by="month", length.out=264)
indiceZ = zoo(datos[,-1], order.by=dt)
```

Pasamos ahora a realizar la descomposición espectral de la serie a través del modelo multiplicativo. Para ello empleamos la función *decompose*:

```
indice_Comp <- decompose(indice, type=c("multiplicative"))
autoplot(indice_Comp, ts.colour = "blue")
```



En el gráfico superior disponemos de cuatro plots diferentes. El primero con nuestra serie temporal, un segundo con los residuos, el tercero con el comportamiento estacional y un último gráfico con la tendencia.

Cabe destacar que el comportamiento estacional se calcula para cada mes como la media de todos los mismos meses observados entre la media de todos los valores observados.

Podemos acceder con las siguientes líneas de código a los componentes que se obtiene de la descomposición espectral y a los coeficientes de estacionalidad:

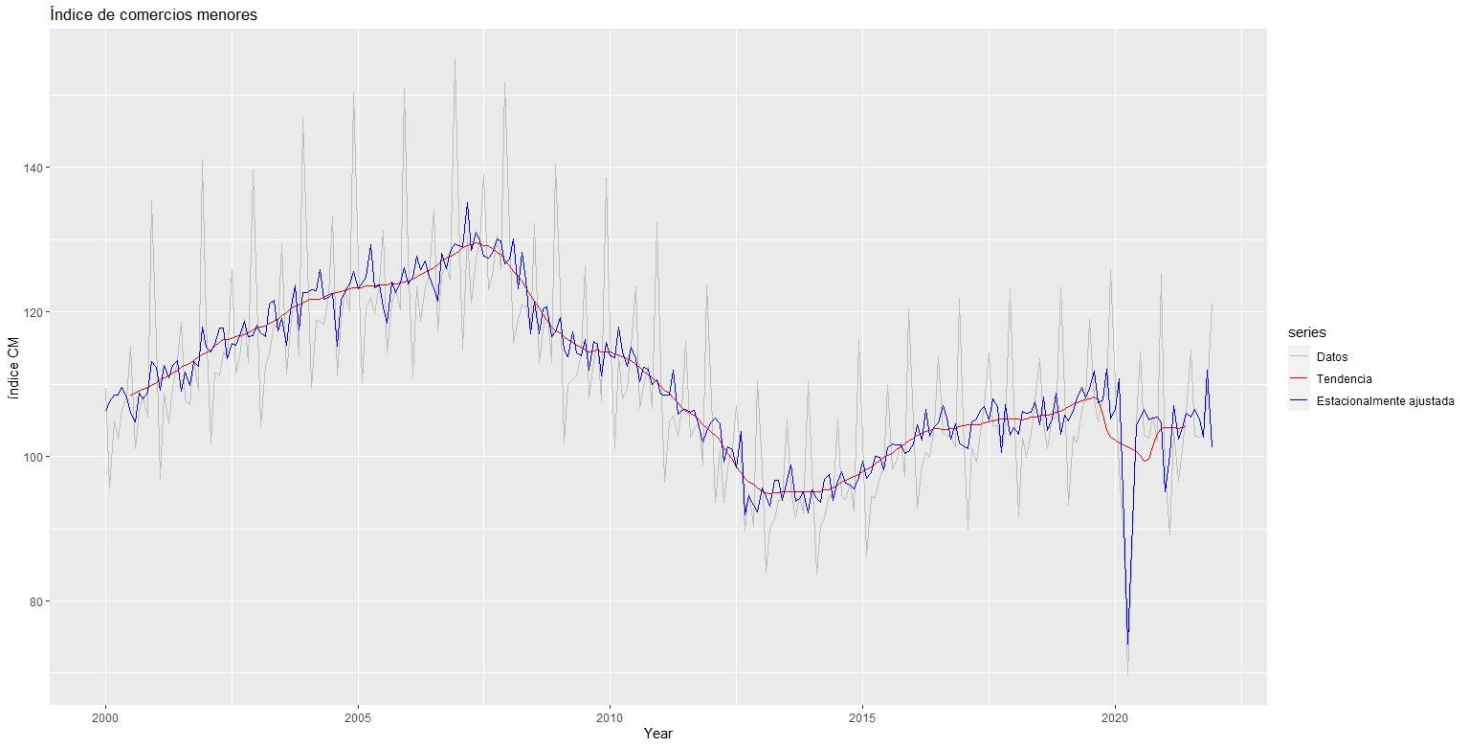
```
print(indice_Comp)

knitr::kable(indice_Comp$figure, digits=2, caption = "Coef Estacionalidad", "simple")
```

Encontramos, por ejemplo, que en el mes de Diciembre el coeficiente de estacionalidad es de 1.20, lo que quiere decir que el índice aumenta un 20% en ese mes con respecto a la media del año.

También puede resultar útil graficar la serie temporal junto con la tendencia y la serie ajustada estacionalmente:

```
autoplot(indice, series="Datos") +
  autolayer(trendcycle(indice_Comp), series="Tendencia") +
  autolayer(seasadj(indice_Comp), series="Estacionalmente ajustada") +
  xlab("Year") + ylab("índice CM") +
```

```
  ggtitle("Índice de comercios menores") +
  scale_colour_manual(values=c("gray", "red", "blue"),
                      breaks=c("Datos", "Tendencia", "Estacionalmente
ajustada"))
```

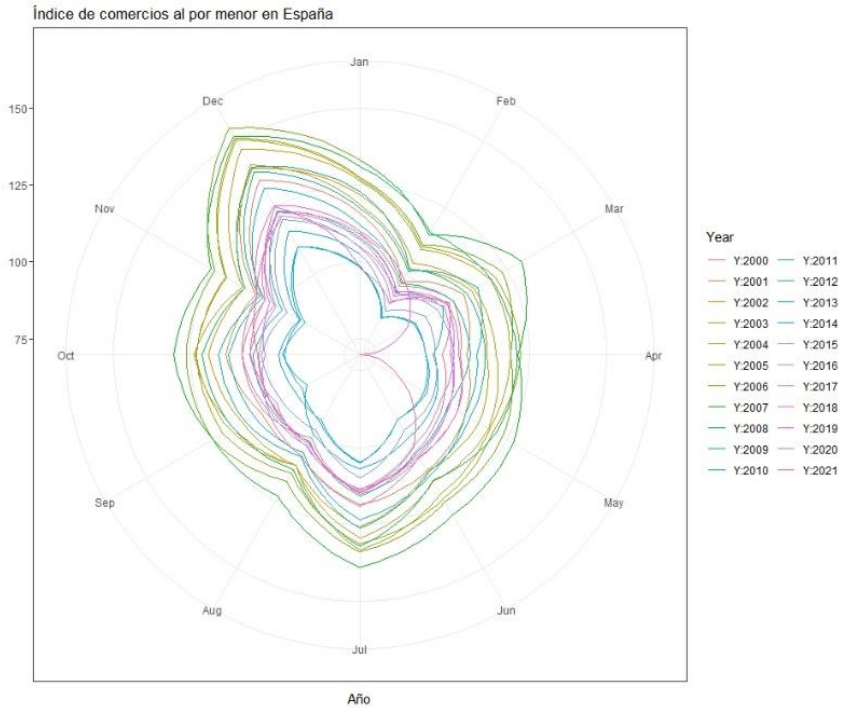


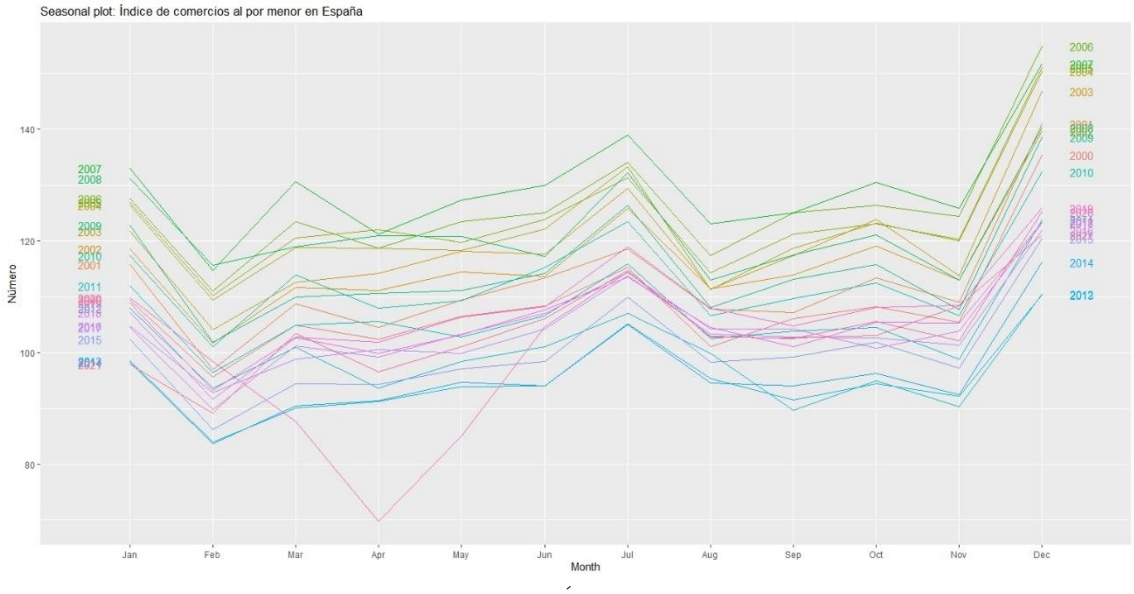
Vemos que los datos estacionalmente ajustados no se ajustan del todo a los originales, esto es porque estamos empleando el mismo coeficiente de estacionalidad para todos los meses. Este coeficiente ha podido ir cambiando con el tiempo, lo que consideraremos con los métodos de suavizado que veremos más adelante.

Para estudiar la estacionalidad resulta útil representar cada año por separado. Primero lo hacemos con coordenadas polares en función de los meses y después en cartesianas:

```
ggseasonplot(indice, polar = TRUE,
             season.labels = c("Jan","Feb","Mar","Apr","May","Jun","Jul",
                               "Aug","Sep","Oct","Nov","Dec")) +
  ggtitle("Índice de comercios al por menor en España") +
  xlab("Año") +
  scale_color_discrete(name= "Year", labels= paste0("Y:",2000:2021)) +
  theme_bw()
```

```
ggseasonplot(indice, year.labels=TRUE, year.labels.left=TRUE) +
  ylab("Número") +
  ggtitle("Seasonal plot: Índice de comercios al por menor en España")
```





Nos llaman la atención algunos datos, como la anomalía del índice en abril del año 2020, el aumento del índice en diciembre y julio, o las depresiones de noviembre y febrero.

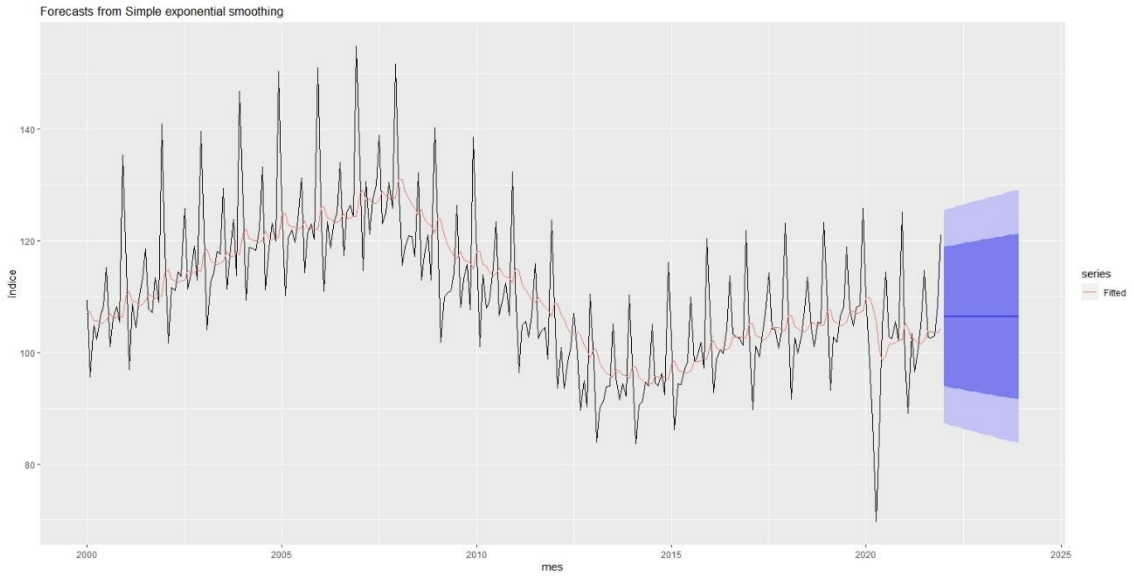
Pasamos ahora a explorar los distintos modelos de suavizado exponencial, pero antes separamos los dos últimos años del dataframe, con los que haremos futuras predicciones:

```
indice_test <- window(indice, start=c(2021,1))
indice_test2 <- window(indice, start=c(2020,1))
```

Para suavizar exponencialmente empleamos la función *ses*. Especificamos la opción Alpha=NULL para minimizar directamente el error cuadrático y predecimos 24 valores, es decir, dos años.

```
indice_s1 <- ses(indice, alpha=NULL, h=24 )
```

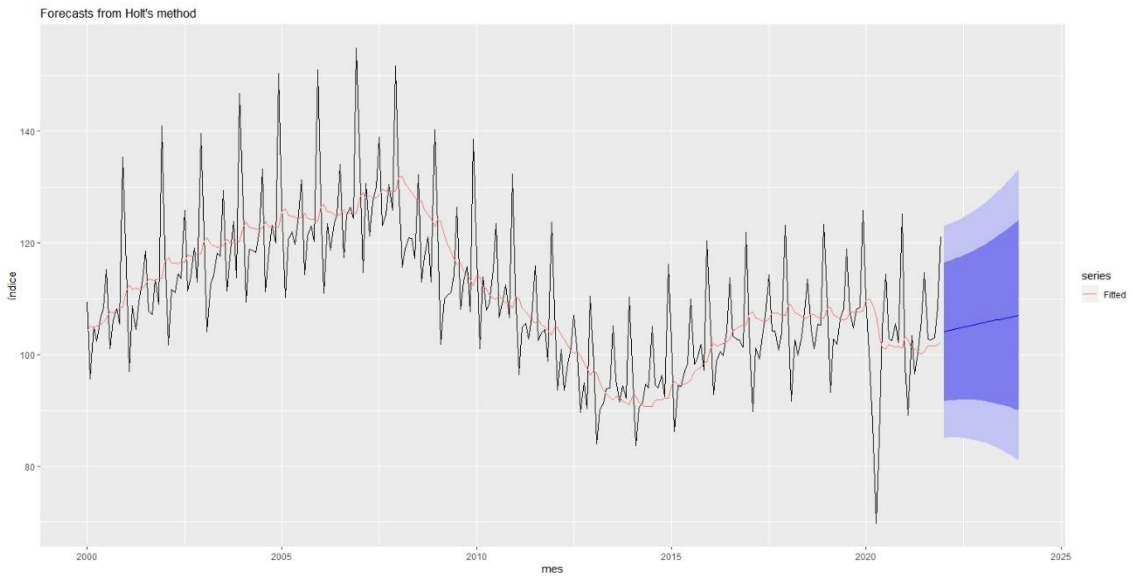
```
autoplot(indice_s1) +
  autolayer(fitted(indice_s1), series = "Fitted") +
  ylab("índice") + xlab("mes")
```



En la gráfica observamos nuestra serie original y en naranja la serie suavizada exponencialmente. Además, en azul tenemos las predicciones junto con los intervalos de confianza del 80% y del 95%. Sabiendo esto, resulta un poco extraño que la predicción de los valores la realice en torno a un eje completamente horizontal. Esto es así porque el modelo no está teniendo en cuenta la tendencia de la serie. Podemos aplicar el método de Holt para incluir la tendencia y mejorar la predicción:

```
indice_s2 <- holt(indice, h=24 )
```

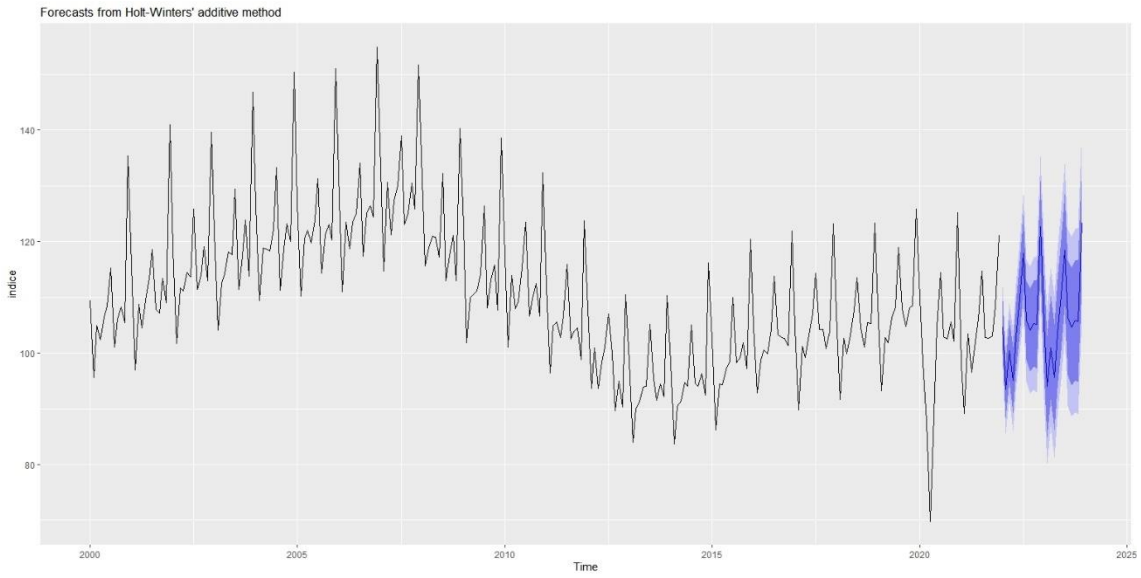
```
autoplot(indice_s2) +
  autolayer(fitted(indice_s2), series = "Fitted") +
  ylab("índice") + xlab("mes")
```



Ahora sí se está siguiendo la tendencia de la serie, por lo que la predicción nos resulta más confiable. Cabe decir que la función *holt* ofrece también la opción de amortiguar la predicción con el input *damped=TRUE*. Con esto conseguimos que las predicciones no se ajusten en torno a una recta, como en los dos casos anteriores, sino en torno a una curva.

Con estos últimos ajustes hemos mejorado un poco nuestro modelo, que se aproxima más a los datos reales. Sin embargo, nos damos cuenta de que no estamos teniendo en cuenta un elemento muy importante de nuestra serie: La estacionalidad. Para conseguir un modelo que que sí incluya la estacionalidad emplearemos el método de *Holt-Winters*, que tiene dos variaciones dependiendo de la naturaleza del componente estacional. Una primera variación sería el método multiplicativo, que se emplea cuando aparecen variaciones en la estacionalidad. La segunda sería la aditiva, que considera constante las variaciones estacionales

```
indice_s3 <- hw(indice, h=24, seasonal="additive", level=c(80,95))
autoplot(indice_s3)
```



Vemos como ahora nuestra predicción de la serie sí muestra un carácter estacional, por lo que el método *Holt-Winters* será el más adecuado. A continuación, generamos una variable *train* que contendrá todos los años excepto el último. Sobre esta variable modelizaremos de cuatro formas diferentes: Aditivo sin amortiguar, multiplicativo sin amortiguar, aditivo amortiguado y multiplicativo amortiguado:

```
train <- subset(indice, end=length(indice) - 12)
fit1 <- hw(train, h=12, seasonal="multiplicative")
fit2 <- hw(train, h=12, seasonal="additive")
fit3 <- hw(train, h=12, seasonal="multiplicative", damped=TRUE)
fit4 <- hw(train, h=12, seasonal="additive", damped=TRUE)
```

Una vez temenos los modelos, extraemos las predicciones y las comparamos con los datos reales:

```
data1 <- as.data.frame(fit1)
prediction1 <- data1[,1]
data2 <- as.data.frame(fit2)
prediction2 <- data2[,1]
data3 <- as.data.frame(fit3)
prediction3 <- data3[,1]
data4 <- as.data.frame(fit4)
prediction4 <- data4[,1]
```

```
values <- window(indice, start=c(2021,1))
values <- as.numeric(values)
```

Una vez que tenemos las predicciones de los cuatro modelos y los valores reales, definimos una función que nos permitirá obtener algunos de los marcadores más relevantes a la hora de medir la calidad de las predicciones de nuestros modelos:

```
getPerformance = function(pred, val) {
  res = pred - val
  MAE = sum(abs(res))/length(val)
  RSS = sum(res^2)
  MSE = RSS/length(val)
  RMSE = sqrt(MSE)
  perf = data.frame(MAE, RSS, MSE, RMSE)
}
```

La aplicamos:

```
params1 <- getPerformance(data1, values)
params2 <- getPerformance(data2, values)
params3 <- getPerformance(data3, values)
params4 <- getPerformance(data4, values)
print(params1)
print(params2)
print(params3)
print(params4)
```

	MAE	RSS	MSE	RMSE
modelo1	39.51	5291.11	440.92	20.99
modelo2	41.43	5789.04	482.42	21.96
modelo3	41.81	5683.07	473.59	21.76
modelo4	41.33	5764.81	480.4	21.92

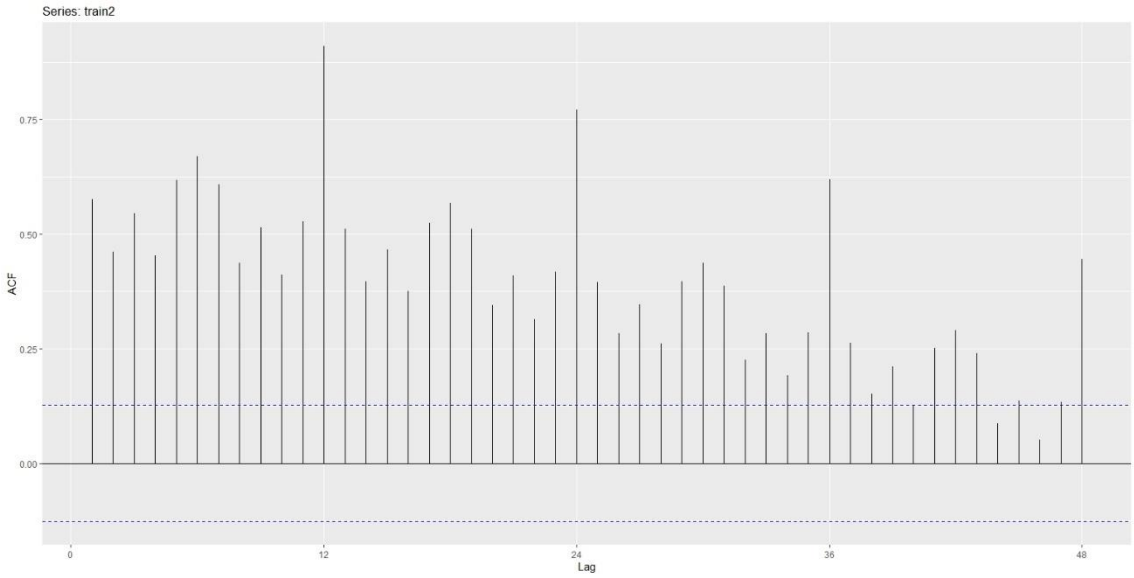
Atendiendo a algunos de los errores decidimos que el modelo más efectivo es el *Holt-Winters* multiplicativo y con la componente estacional sin amortiguar.

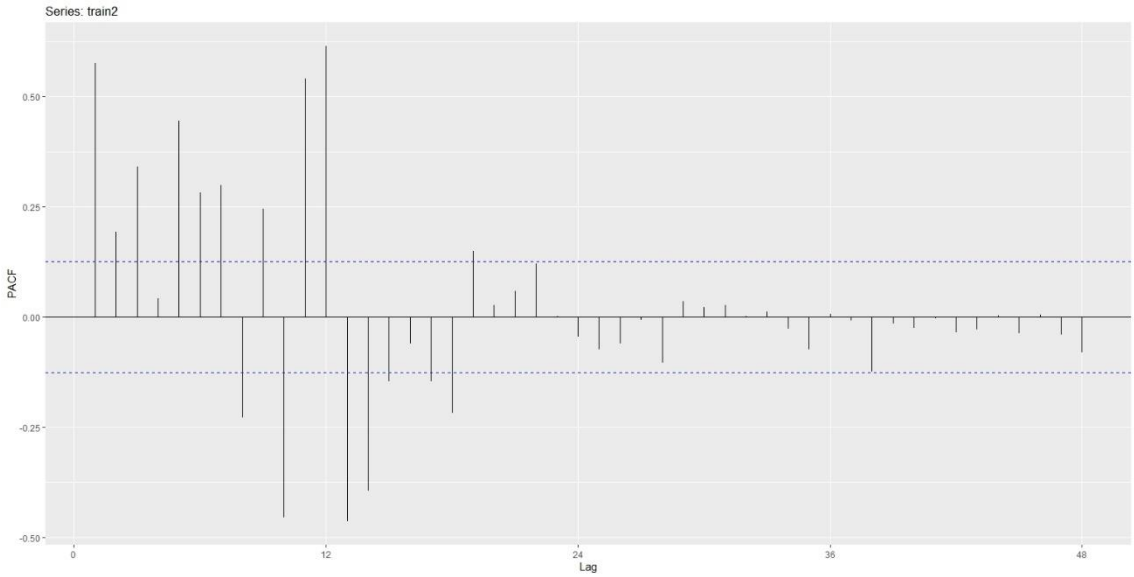
Pasamos ahora a estudiar los correlogramas de nuestra serie temporal, a la que le vamos a quitar los dos últimos años para eliminar la anomalía de la pandemia, que hemos decidido quitarla para que los ajustes salgan con una mayor calidad y para después poder predecir sobre esos meses.

train2 <- subset(indice, end=length(indice) - 24)

ggAcf(train2, 48) #Correlaciones

ggPacf(train2, 48) #Correlaciones parciales

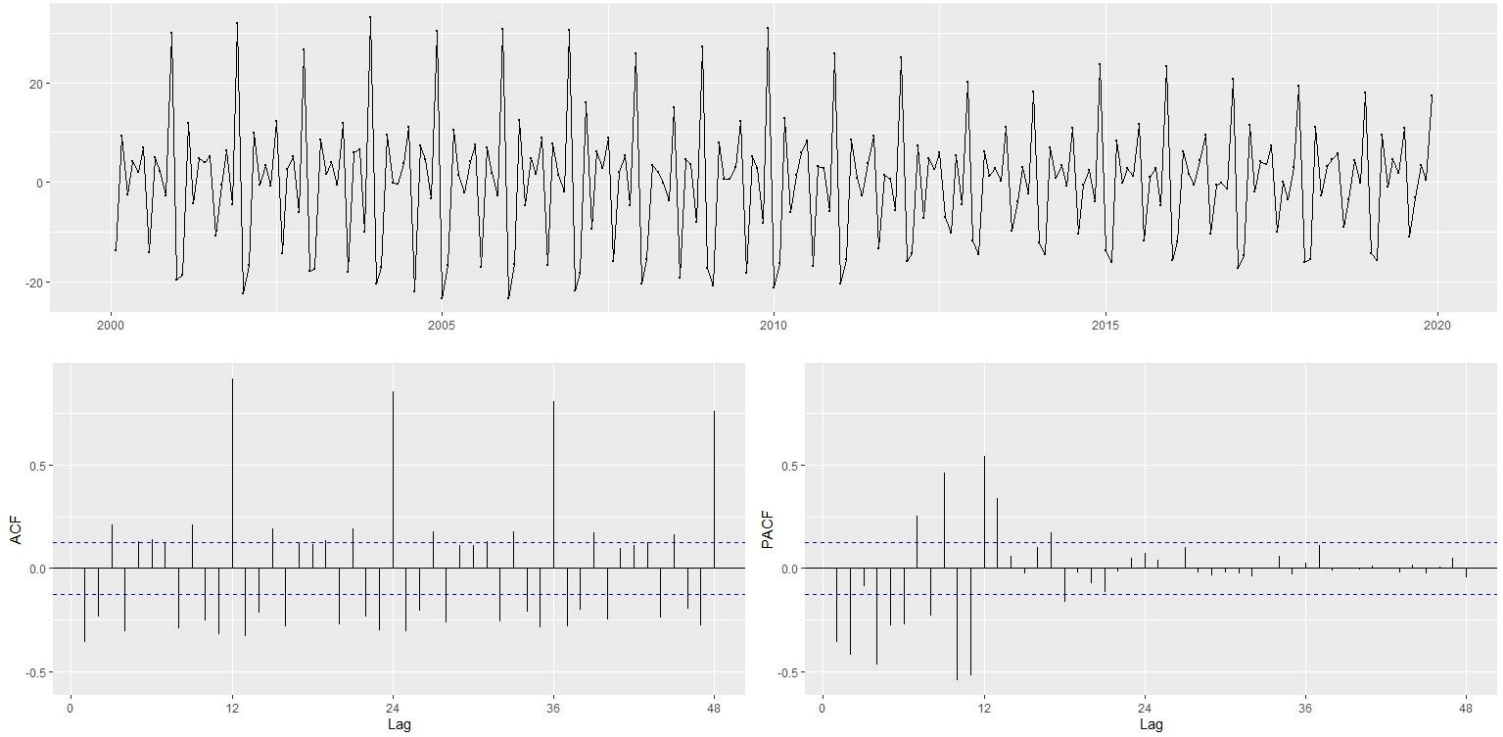




En la primera imagen observamos las autocorrelaciones decrecen de forma continuada, lo que nos indica la falta de estacionariedad de nuestra serie, lo que ya se podía saber mirando simplemente la serie representada. Esto lo modelaremos a continuación diferenciándola.

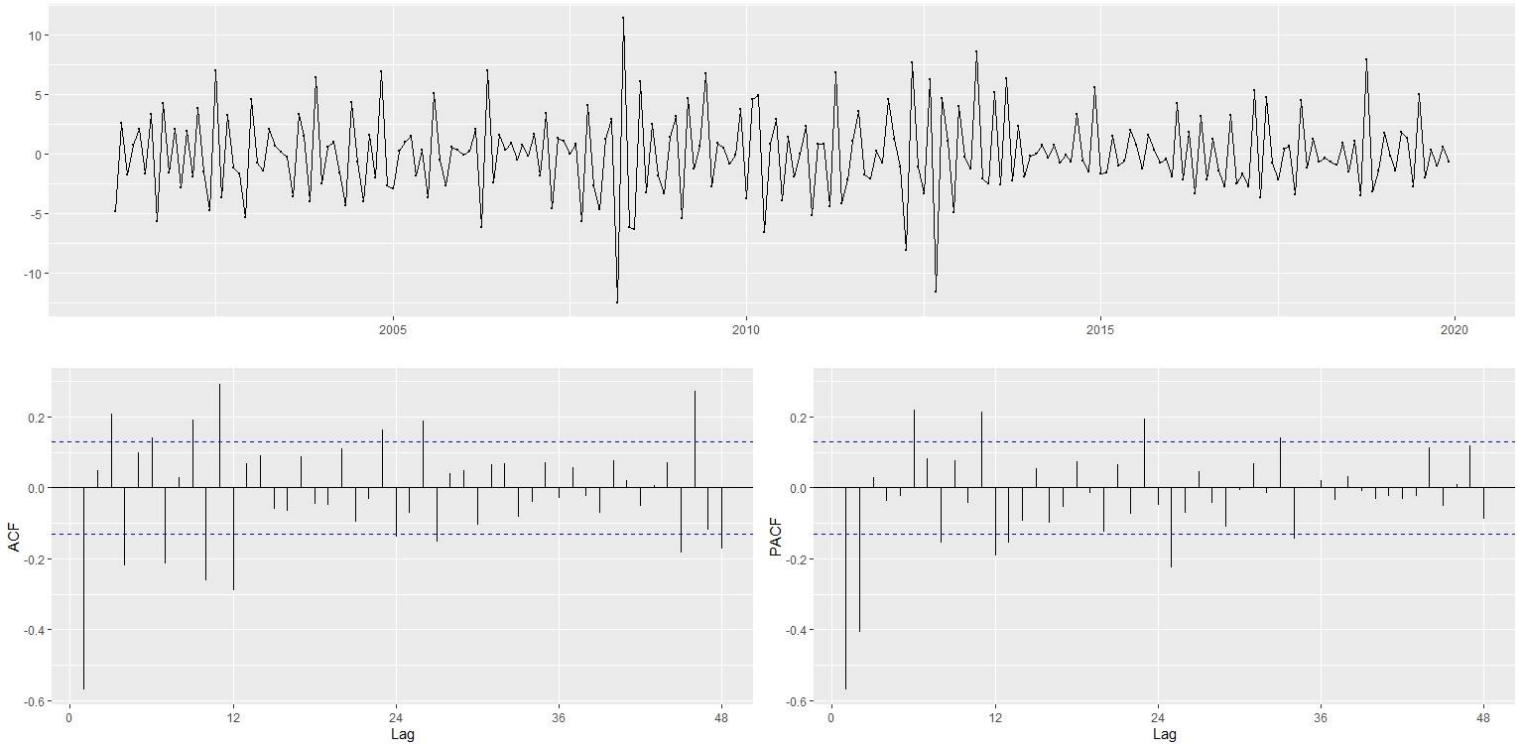
Con respecto a las autocorrelaciones parciales, vemos que tenemos algunas muy fuertes (índices, 1, 11 y 12) con valores superiores a 0.5. Otras no superan esa cota, pero sí las bandas de confianza. En cualquier caso, lo modelizaremos más adelante, primero nos encargamos de la estacionariedad:

ggtsdisplay(diff(train2), lag=48)



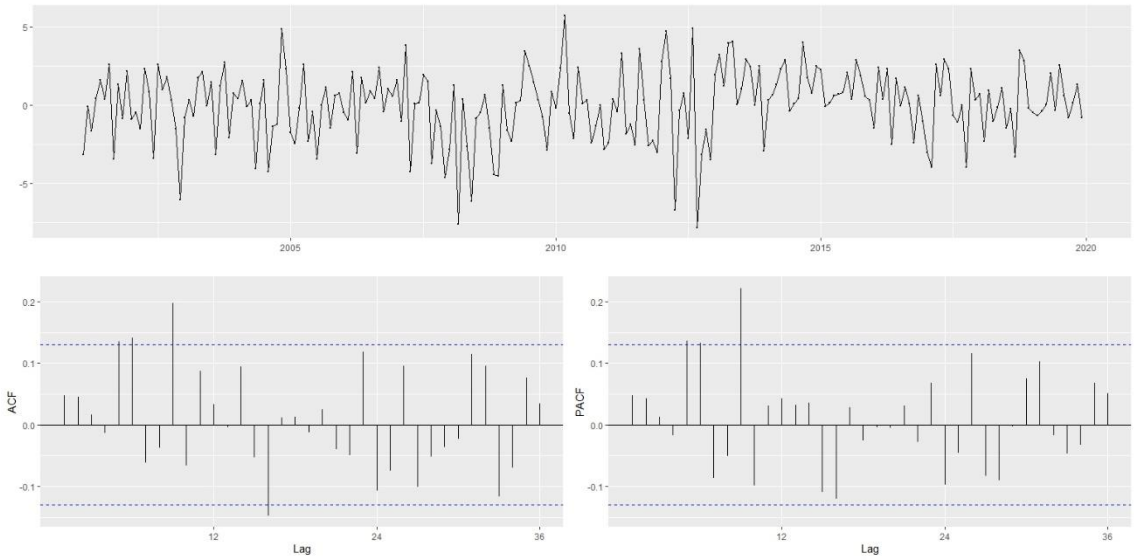
Vemos en el gráfico de la serie que hemos ganado bastante estacionariedad, tanto con respecto a la media como con respecto a la varianza de los datos. Esto también se refleja en el correlograma, que ya no decrece de forma continuada, sino que lo hace de forma abrupta. Podemos centrarnos entonces en la parte estacional tan marcada que nos muestra también el primer correlograma. Para ello diferenciamos, pero ahora estacionalmente (orden 12):

```
ggtsdisplay(diff(diff(train2),12), lag=48)
```



Se observa a simple vista cómo la serie temporal tiene un comportamiento estacional más difícil de ver. Esta información también nos la dan los gráficos inferiores, que ya no muestran autocorrelaciones tan marcadas y regulares cada doce meses. Buscamos ahora eliminar la correlación parcial tan grande que aparece hasta orden dos. Para ello volvemos a diferenciar nuestra serie, pero usamos ahora la función *Arima*, que nos permite especificar de una forma más simple si queremos diferenciar la parte regular o la estacional, y sobre qué componente hacerlo. Como hemos dicho, diferenciamos hasta orden dos la parte regular autorregresiva y también incluimos un uno en la parte de medias móviles estacional, pues sigue habiendo una autocorrelación importante en el orden doce:

```
serie_dif <- diff(diff(train2), 12)

serie_dif %>% Arima(order=c(2,0,0), seasonal=c(0,0,1)) %>% residuals() %>%
ggtsdisplay()
```

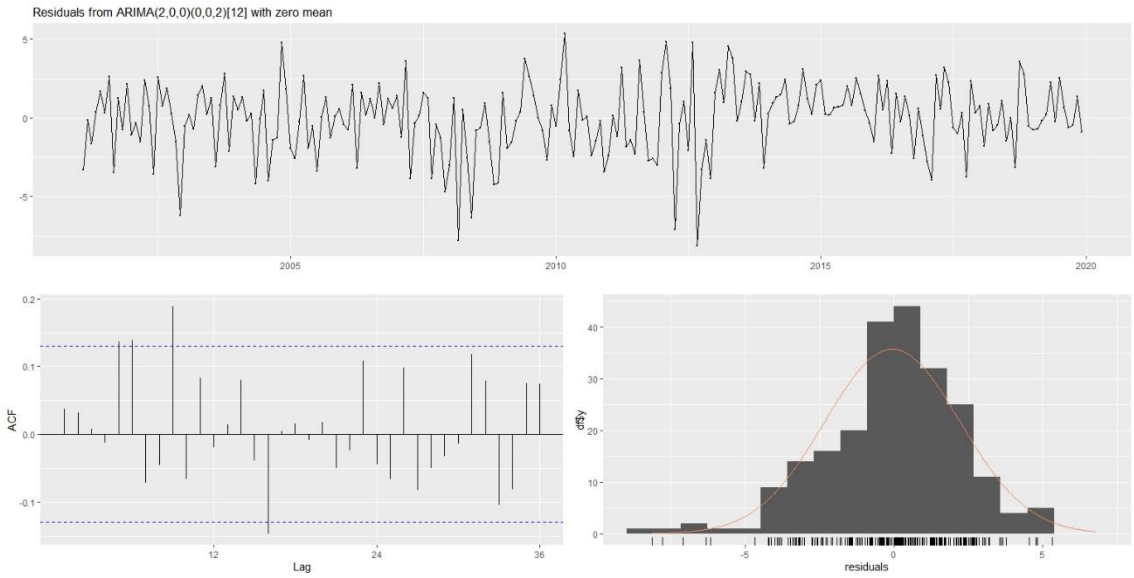


Obtenemos correlogramas más aceptables, que no presentan valores altos para los primeros índices ni tampoco nos muestran la presencia de estacionalidad. En el gráfico de las correlaciones parciales, sólo el noveno índice supera la banda de confianza, superando el valor 0.2. Esto quizás deberíamos asumirlo, ya que no es recomendable llegar a índices tan elevados para los modelos ARIMA.

Nuestro análisis manual podemos comprobarlo con la elección automática del modelo óptimo a través de la función *auto.arima()*. Al pasarle como input la serie temporal que teníamos definida como *serie_dif* nos recomienda la elección de un ARIMA con coeficientes (2,0,0)(0,0,2), es decir que diferencia doblemente la parte de medias móviles de la parte estacional, cuando nosotros lo hemos hecho una sola vez. A pesar de que los correlogramas son muy parecidos y que el modelo que nos proponen sigue sin resolver el problema con el índice 9, optamos por quedarnos con él para mejorar el ajuste (presenta menor AIC), aunque sea un poco más complejo. Tenemos entonces que nuestro ajuste ARIMA total de la serie temporal original tiene los coeficientes: (2,1,0)(0,1,2)

Después de generar el modelo mostramos sus parámetros y sus residuos:

```
model_auto <- auto.arima(serie_dif, seasonal=TRUE)

checkresiduals(model_auto)

print(model_auto)
```




```
data:  Residuals from ARIMA(2,0,0)(0,0,2)[12] with zero mean
Q* = 34.215, df = 20, p-value = 0.02471

Model df: 4.   Total lags used: 24
```

```
Series: serie_dif
ARIMA(2,0,0)(0,0,2)[12] with zero mean

Coefficients:
          ar1      ar2     sma1     sma2
      -0.7943  -0.4288  -0.4603  -0.1119
s.e.   0.0606   0.0617   0.0712   0.0768

sigma^2 = 5.248:  log likelihood = -510.66
AIC=1031.31   AICc=1031.58   BIC=1048.44
```

Los residuos ideales deben tener media 0, varianza constante y ausencia de correlación para cualquier retardo. Vemos en el primer gráfico que las condiciones de la media y la varianza se cumplen aproximadamente. Sin embargo, encontramos correlaciones que sobrepasan las bandas de confianza para los retardos con índice 9 y 16, indicando que el ajuste está lejos de ser perfecto. Por otra parte, el p-valor obtenido es mayor a 0.01, lo que acepta la hipótesis de que los residuos estén incorrelados.

Una vez tenemos nuestro modelo pasamos a hacer previsiones de dos años, que eran los datos que habíamos reservado anteriormente. Las mostramos junto con los intervalos de confianza:

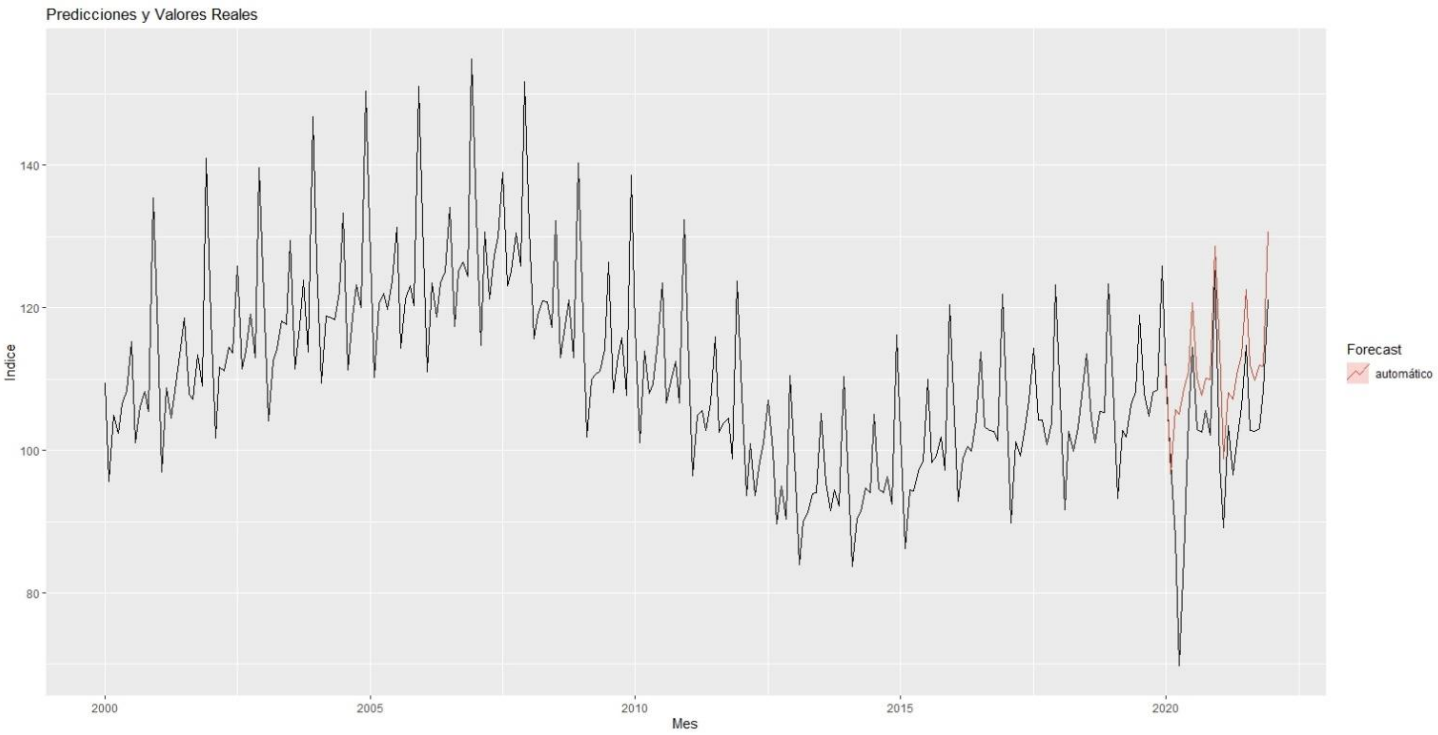
```
pred <- forecast(model_auto, h=24)
```

```
cbind("prediccion" = pred$mean,
      "L80" = pred$lower,
      "U80" = pred$upper) %>% knitr::kable(caption="Predicciones", "simple")
```

Table: Predicciones

prediccion	L80.80%	L80.95%	U80.80%	U80.95%
112.03922	109.10329	107.54910	114.97515	116.5293
96.73048	93.73309	92.14637	99.72787	101.3146
105.74836	102.52068	100.81206	108.97604	110.6847
105.07267	101.41251	99.47494	108.73283	110.6704
108.64441	104.83619	102.82024	112.45263	114.4686
111.06011	107.01615	104.87541	115.10407	117.2448
120.72653	116.44568	114.17954	125.00738	127.2735
110.03717	105.58117	103.22231	114.49317	116.8520
107.71524	103.05890	100.59398	112.37159	114.8365
110.07728	105.23410	102.67028	114.92045	117.4843
109.96969	104.95551	102.30116	114.98386	117.6382
128.60218	123.41375	120.66716	133.79062	136.5372
114.02272	108.07661	104.92892	119.96884	123.1165
98.81909	92.65056	89.38513	104.98762	108.2530
108.08996	101.61418	98.18611	114.56574	117.9938
107.14032	100.28450	96.65525	113.99613	117.6254
110.57152	103.46069	99.69644	117.68236	121.4466
113.24156	105.83943	101.92098	120.64369	124.5621
122.48120	114.79294	110.72302	130.16945	134.2394
111.95089	104.00972	99.80592	119.89206	124.0959
109.75439	101.55286	97.21123	117.95592	122.2976
112.00036	103.54907	99.07523	120.45165	124.9255
111.77865	103.08899	98.48897	120.46831	125.0683
130.65095	121.72499	116.99986	139.57692	144.3020

Graficamos:



Podemos finalmente aplicar la función definida con anterioridad para obtener los errores:

```
values <- window(indice, start=c(2020,1))
values <- as.numeric(values)

params <- getPerformance(as.numeric(pred$mean), values)
```

```
       MAE      RSS      MSE     RMSE
1 9.370708 3390.452 141.2688 11.88566
```

Observamos en el gráfico anterior que los datos se ajustan con bastante precisión a los valores reales, exceptuando los primeros meses del año 2020, donde llega incluso a predecir un índice treinta veces mayor que el real. Esto era algo que esperábamos, pues la crisis del coronavirus fue un suceso sin precedentes, imprevisible y que tuvo un impacto enorme en la sociedad, por lo que ningún modelo puede adaptarse a un fenómeno de tal naturaleza. La crisis del coronavirus afectó al ser humano en todos los sentidos posibles y el sector económico minorista en España no es una excepción. Los modelos predicen, sí, pero sobre un histórico de datos. A pesar de todo obtenemos un RMSE igual a 11.89, que es una cifra más que aceptable teniendo en cuenta que los datos orbitan en torno a valores de 120 y que intentamos predecir sobre un *outlier* como son los primeros meses de 2020.