

MODELACIÓN DEL PRECIO PARA LA COMPRA Y VENTA DE ACEITE DE SOYA

Nidia Munevar - Leonardo Palacios

2023-10-02

Contents

1	Resumen	5
2	Introduccion	7
3	Justificacion	9
4	Serie de Tiempo	11
5	Analisis Exploratorio	15
6	Promedio Movil	19
7	Rezago	23
8	Descomposicion	25
9	Estacionariedad	29
10	Diferenciacion	31
11	Holter-Winter	33
12	ARIMA	45
13	Prophet	49
14	Elman	55
15	Jordan	59

Chapter 1

Resumen

El proyecto aplicado a realizar es la modelación del precio para la compra y venta de aceite de soya.

Chapter 2

Introduccion

En el mercado de venta y compra de materias primas agrícolas intervienen diferentes actores, los precios son públicos y son afectados por diferentes variables tales como el precio del petróleo, la tasa de cambio, el clima entre otros elementos. La necesidad de los actores es mejorar sus decisiones y de esta forma su rentabilidad, los precios de las materias primas afectan directamente al mercado y a los precios de los bienes producidos a partir de estas, es decir estos valores terminan impactando al comprador final.

Chapter 3

Justificacion

El proyecto está planteado ante una necesidad de los actores que requieren mejorar sus decisiones y de esta forma su rentabilidad. Los precios de las materias primas afectan directamente al mercado y a los precios de los bienes producidos a partir de estas materias, es decir estos valores terminan impactando al comprador final.

Chapter 4

Serie de Tiempo

```
# Cargar la biblioteca quantmod  
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##  
## Attaching package: 'zoo'
```

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

```
## Loading required package: TTR
```

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

```
# Especificar el símbolo para futuros de soja  
symbol <- "ZS=F"
```

```
# Descargar los datos históricos desde el 1 de enero de 2010 hasta hoy  
getSymbols(symbol, from = "2010-01-01", to = Sys.Date(), auto.assign = TRUE)
```

```
## [1] "ZS=F"
```

```
# Crear un data frame con la serie de tiempo
soybean_data <- data.frame(Date = index(get(symbol)),
                           Open = Op(get(symbol)),
                           High = Hi(get(symbol)),
                           Low = Lo(get(symbol)),
                           Close = Cl(get(symbol)),
                           Volume = Vo(get(symbol))
                           )

# Eliminar filas con valores NA
soybean_data <- na.omit(soybean_data)

# Muestra los primeros registros del data frame
head(soybean_data)
```

```
##           Date ZS.F.Open ZS.F.High ZS.F.Low ZS.F.Close ZS.F.Volume
## 2010-01-04 2010-01-04   1043.00   1065.50  1041.25   1049.50       25947
## 2010-01-05 2010-01-05   1047.00   1056.00  1042.00   1052.25       21073
## 2010-01-06 2010-01-06   1050.00   1058.50  1042.75   1050.50       17567
## 2010-01-07 2010-01-07   1050.50   1052.00  1016.50   1017.75       11750
## 2010-01-08 2010-01-08   1018.25   1018.25  1005.00   1013.00       11750
## 2010-01-11 2010-01-11   1014.00   1022.00   997.50   1001.75       11750
```

```
class(soybean_data)
```

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

```
# Cargar la biblioteca xts
library(xts)

# Crear una serie de tiempo xts a partir del data frame soybean_data
soybean_xts <- xts(soybean_data[, -1], order.by = soybean_data$Date)

# Verificar la serie de tiempo
head(soybean_xts)
```

```
##           ZS.F.Open ZS.F.High ZS.F.Low ZS.F.Close ZS.F.Volume
## 2010-01-04   1043.00   1065.50  1041.25   1049.50       25947
## 2010-01-05   1047.00   1056.00  1042.00   1052.25       21073
## 2010-01-06   1050.00   1058.50  1042.75   1050.50       17567
## 2010-01-07   1050.50   1052.00  1016.50   1017.75       11750
## 2010-01-08   1018.25   1018.25  1005.00   1013.00       11750
## 2010-01-11   1014.00   1022.00   997.50   1001.75       11750
```

```
class(soybean_xts)
```

```
## [1] "xts" "zoo"
```


Chapter 5

Analisis Exploratorio

```
# Cargar la biblioteca quantmod
library(quantmod)

# Especificar el símbolo para futuros de soja
symbol <- "ZS=F"

# Descargar los datos históricos desde el 1 de enero de 2010 hasta hoy
getSymbols(symbol, from = "2010-01-01", to = Sys.Date(), auto.assign = TRUE)
```

```
## [1] "ZS=F"
```

```
# Crear un data frame con la serie de tiempo
soybean_data <- data.frame(Date = index(get(symbol)),
                           Open = Op(get(symbol)),
                           High = Hi(get(symbol)),
                           Low = Lo(get(symbol)),
                           Close = Cl(get(symbol)),
                           Volume = Vo(get(symbol))
)
```

```
# Eliminar filas con valores NA
soybean_data <- na.omit(soybean_data)
```

```
head(soybean_data)
```

```
##           Date ZS.F.Open ZS.F.High ZS.F.Low ZS.F.Close ZS.F.Volume
```

```
## 2010-01-04 2010-01-04 1043.00 1065.50 1041.25 1049.50 25947
## 2010-01-05 2010-01-05 1047.00 1056.00 1042.00 1052.25 21073
## 2010-01-06 2010-01-06 1050.00 1058.50 1042.75 1050.50 17567
## 2010-01-07 2010-01-07 1050.50 1052.00 1016.50 1017.75 11750
## 2010-01-08 2010-01-08 1018.25 1018.25 1005.00 1013.00 11750
## 2010-01-11 2010-01-11 1014.00 1022.00 997.50 1001.75 11750
```

```
# Cargar la biblioteca xts
library(xts)

# Crear una serie de tiempo xts a partir del data frame soybean_data
soybean_xts <- xts(soybean_data[, -1], order.by = soybean_data$Date)

# Verificar la serie de tiempo
head(soybean_xts)
```

```
##           ZS.F.Open ZS.F.High ZS.F.Low ZS.F.Close ZS.F.Volume
## 2010-01-04    1043.00    1065.50    1041.25    1049.50        25947
## 2010-01-05    1047.00    1056.00    1042.00    1052.25        21073
## 2010-01-06    1050.00    1058.50    1042.75    1050.50        17567
## 2010-01-07    1050.50    1052.00    1016.50    1017.75        11750
## 2010-01-08    1018.25    1018.25    1005.00    1013.00        11750
## 2010-01-11    1014.00    1022.00     997.50    1001.75        11750
```

```
class(soybean_xts)
```

```
## [1] "xts" "zoo"
```

```
# Acceder a la columna "ZS.F.Close" en soybean_xts
close_prices <- soybean_xts[, "ZS.F.Close"]

# Imprimir las primeras filas de la columna Close
print(head(close_prices))
```

```
##           ZS.F.Close
## 2010-01-04    1049.50
## 2010-01-05    1052.25
## 2010-01-06    1050.50
## 2010-01-07    1017.75
## 2010-01-08    1013.00
## 2010-01-11    1001.75
```



```
head(soybean_xts)
```

```
##           ZS.F.Open ZS.F.High ZS.F.Low ZS.F.Close ZS.F.Volume
## 2010-01-04   1043.00   1065.50  1041.25   1049.50       25947
## 2010-01-05   1047.00   1056.00  1042.00   1052.25       21073
## 2010-01-06   1050.00   1058.50  1042.75   1050.50       17567
## 2010-01-07   1050.50   1052.00  1016.50   1017.75       11750
## 2010-01-08   1018.25   1018.25  1005.00   1013.00       11750
## 2010-01-11   1014.00   1022.00   997.50   1001.75       11750
```

```
# Cargar la biblioteca ggplot2 para hacer gráficos
```

```
library(ggplot2)
```

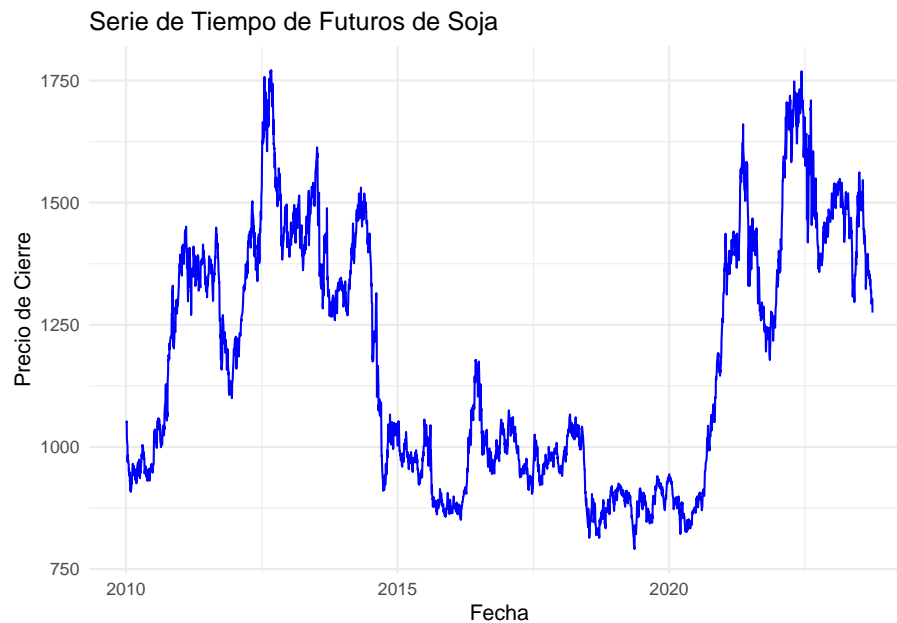
```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
# Crear un gráfico de serie de tiempo
```

```
ggplot(data = NULL, aes(x = index(close_prices), y = close_prices)) +  
  geom_line(color = "blue") +  
  labs(x = "Fecha", y = "Precio de Cierre", title = "Serie de Tiempo de Futuros de Soja") +  
  theme_minimal()
```

```
## Don't know how to automatically pick scale for object of type <xts/zoo>.
```

```
## Defaulting to continuous.
```



Chapter 6

Promedio Movil

Cuando aplicamos un promedio móvil a una serie de tiempo, cada punto de la serie transformada (promediada) es el promedio de un número determinado de puntos anteriores, actuales y futuros de la serie original. Este número de puntos que decides promediar se llama “ventana” del promedio móvil.

```
# Cargar la biblioteca quantmod
library(quantmod)

# Especificar el símbolo para futuros de soja
symbol <- "ZS=F"

# Descargar los datos históricos desde el 1 de enero de 2010 hasta hoy
getSymbols(symbol, from = "2010-01-01", to = Sys.Date(), auto.assign = TRUE)
```

```
## Warning: ZS=F contains missing values. Some functions will not work if objects
## contain missing values in the middle of the series. Consider using na.omit(),
## na.approx(), na.fill(), etc to remove or replace them.
```

```
## [1] "ZS=F"
```

```
# Crear un data frame con la serie de tiempo
soybean_data <- data.frame(Date = index(get(symbol)),
                           Open = Op(get(symbol)),
                           High = Hi(get(symbol)),
                           Low = Lo(get(symbol)),
                           Close = Cl(get(symbol)),
                           Volume = Vo(get(symbol))
                           )
```

```
# Eliminar filas con valores NA
soybean_data <- na.omit(soybean_data)
```

```
# Muestra los primeros registros del data frame
# head(soybean_data)
```

```
# Cargar la biblioteca xts
library(xts)
```

```
# Crear una serie de tiempo xts a partir del data frame soybean_data
soybean_xts <- xts(soybean_data[, -1], order.by = soybean_data$Date)
```

```
# Verificar la serie de tiempo
# head(soybean_xts)
```

```
library(ggplot2)
library(TTR)
library(scales)
```

```
## Warning: package 'scales' was built under R version 4.2.3
```

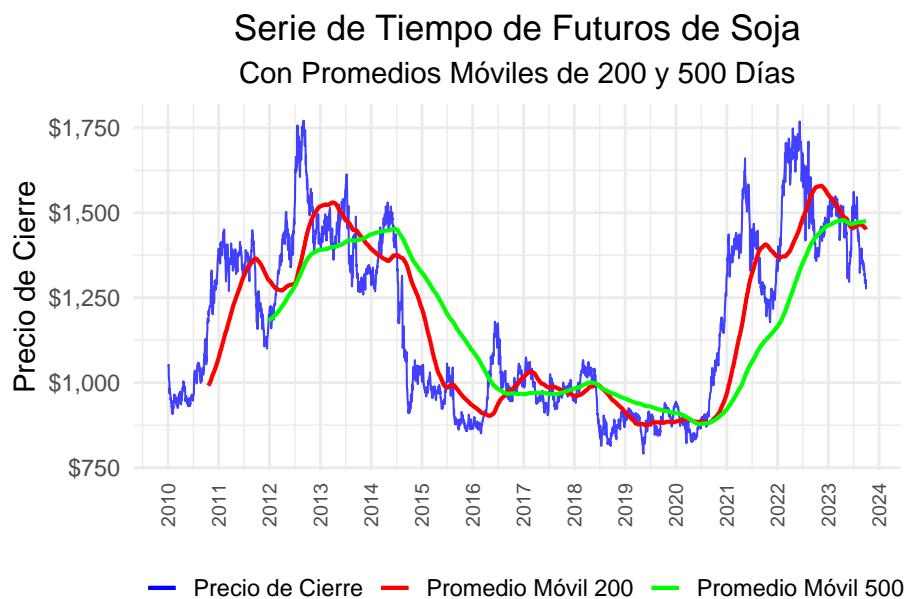
```
# Convertir el objeto xts a data.frame
soybean_df <- as.data.frame(soybean_xts)
soybean_df$Date <- index(soybean_xts)
```

```
# Calcular SMA_200 y SMA_500
soybean_df$SMA_200 <- SMA(soybean_df$ZS.F.Close, n = 200)
soybean_df$SMA_500 <- SMA(soybean_df$ZS.F.Close, n = 500)
```

```
# Usar ggplot2 para visualizar los datos
ggplot(soybean_df, aes(x = Date)) +
  geom_line(aes(y = ZS.F.Close, color = 'Precio de Cierre'), alpha = 0.75) +
  geom_line(aes(y = SMA_200, color = 'Promedio Móvil 200'), size = 1, na.rm = TRUE) +
  geom_line(aes(y = SMA_500, color = 'Promedio Móvil 500'), size = 1, na.rm = TRUE) +
  theme_minimal(base_size = 15) +
  labs(title = 'Serie de Tiempo de Futuros de Soja',
        subtitle = 'Con Promedios Móviles de 200 y 500 Días',
        y = 'Precio de Cierre') +
  theme(axis.title.x = element_blank(),
        axis.text.x = element_text(size = 10, angle = 90, vjust = 0.5),
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5),
        legend.position = "bottom") +
```

```
scale_x_date(date_breaks = "1 year", date_labels = "%Y") +
scale_y_continuous(labels = dollar_format()) +
scale_color_manual(values = c('Precio de Cierre' = 'blue', 'Promedio Móvil 200' = 'red', 'Promedio Móvil 500' = 'green'),
name = "")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Observaciones:

Entre los años 2013 y mediados del 2014 se puede ver cambios en la tendencia de la serie de tiempo de futuros de la soja, tanto para el promedio móvil de 200 días, como para el de 500 días el cual es mas marcado.

Entre los años 2021 y mediados del 2023 se puede ver cambios en la tendencia de la serie de tiempo de futuros de la soja, tanto para el promedio móvil de 200 días, como para el de 500 días el cual es mas marcado. Se podría llegar a validar por medio de un mayor estudio de este tiempo si la afectación fue causada por el desarrollo de la pandemia del covid-19 la cual inicio en marzo de 2020 e inicio a retrocer en Agosto de 2021 cuando se inicio el uso de las vacunas.

Al suavizar las fluctuaciones menores, a través de los promedios móviles se logro resaltar las tendencias subyacentes en los datos.

Chapter 7

Rezago

El concepto de rezago es fundamental para analizar y modelar series de tiempo porque permite entender cómo los valores pasados pueden influir en los valores presentes o futuros de la serie. Al analizar los rezagos, podemos identificar patrones, hacer predicciones más precisas y entender mejor la dinámica subyacente de los datos.

```
# Sin cargar la librería dplyr para evitar conflictos
datos_lag <- data.frame(
  Close = as.numeric(coredata(soybean_xts)),
  Lag = as.numeric(coredata(stats::lag(soybean_xts))) # Usar stats::lag para evitar conflictos
)

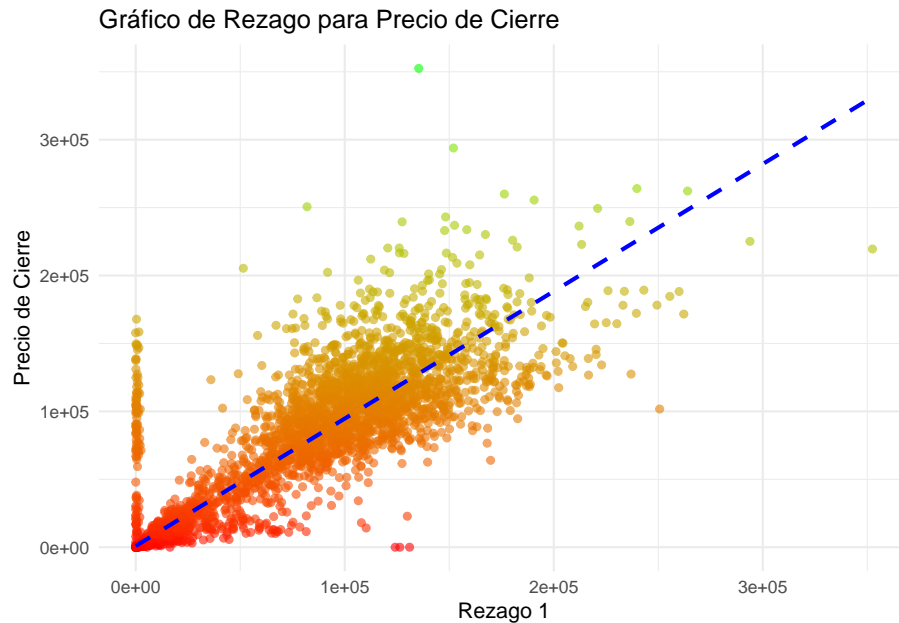
# Comprobando que ambos vectores tengan la misma longitud
stopifnot(length(datos_lag$Close) == length(datos_lag$Lag)) # Detiene la ejecución si no son TRUE

# Crear el gráfico de rezago con ggplot2
library(ggplot2)
ggplot(datos_lag, aes(x=Lag, y=Close)) +
  geom_point(aes(color = Close), alpha=0.6) +
  geom_smooth(method = 'lm', se = FALSE, color="blue", linetype="dashed") +
  scale_color_gradient(low="red", high="green") +
  theme_minimal() +
  labs(title="Gráfico de Rezago para Precio de Cierre", x="Rezago 1", y="Precio de Cierre") +
  theme(legend.position="none")

## `geom_smooth()` using formula = 'y ~ x'

## Warning: Removed 5 rows containing non-finite values (`stat_smooth()`).
```

```
## Warning: Removed 5 rows containing missing values (`geom_point()`).
```



```
#library(forecast)
#class(soybean_xts)
#as.ts(soybean_xts)
#modelo<-auto.arima(soybean_xts)
#modelo

#soybean_tsp <- diff(soybean_xts)
#print(soybean_tsp)
```

Se ve un patrón claro o una agrupación de puntos en el gráfico de rezago 1, por lo tanto es probable que exista una autocorrelación significativa. Se puede considerar modelos de series de tiempo como ARIMA que toman en cuenta la autocorrelación para hacer predicciones más precisas de ser necesario.

Chapter 8

Descomposicion

Se refiere a los patrones o tendencias que se repiten a intervalos regulares, como cada día, mes, trimestre o año, dependiendo de la frecuencia de los datos. En otras palabras, es como un ciclo que se repite en el tiempo.

DESCOMPOSICION

Con la funcion decompose podemos hallar la:

Observed: Serie de tiempo original.

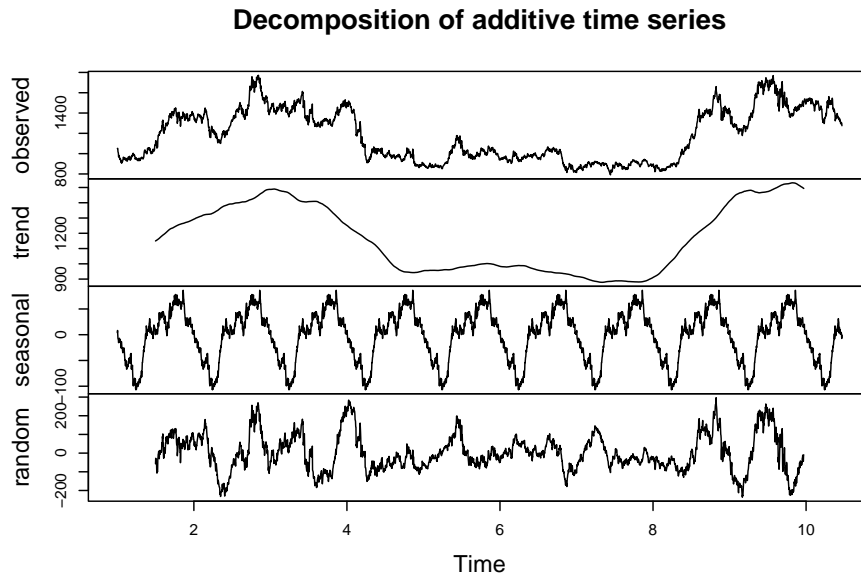
Tendencia (trend): Muestra la dirección general en la que se mueven los datos a largo plazo, sin tener en cuenta las fluctuaciones estacionales o irregulares.

Estacionalidad (seasonal): Representa las fluctuaciones que ocurren en intervalos regulares, como los cambios diarios, mensuales o anuales, debido a la estacionalidad.

Error o Residuo (random): Es la parte de la serie de tiempo que no se puede atribuir ni a la tendencia ni a la estacionalidad. Captura la variabilidad en los datos que no se puede explicar por los otros dos componentes.

```
# Convertir a objeto ts, aquí supondré que tienes datos diarios.
frecuencia <- 365 # (12 para mensual, 4 para trimestral, etc.)
soybean_ts <- ts(soybean_xts[, "ZS.F.Close"], frequency = frecuencia)

# Utilizar decompose
soybean_decomposed <- decompose(soybean_ts)
plot(soybean_decomposed)
```



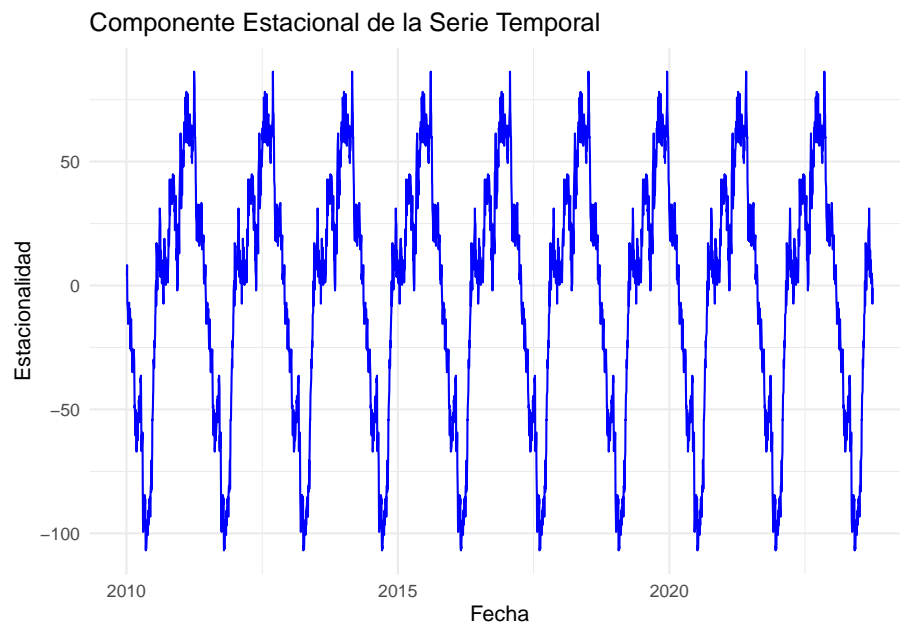
Al validar el componente estacional de la serie por separado, como se ve en la siguiente imagen:

```
# Extraer el componente estacional y convertir el tiempo en fecha
seasonal_df <- data.frame(
  Date = as.Date(index(soybean_xts)),
  Seasonal = as.numeric(soybean_decomposed$seasonal)
)
```

```
# Eliminar las filas con NA en el componente estacional (pueden aparecer dependiendo d
seasonal_df <- seasonal_df[!is.na(seasonal_df$Seasonal), ]
```

```
library(ggplot2)

ggplot(seasonal_df, aes(x=Date, y=Seasonal)) +
  geom_line(color="blue") +
  theme_minimal() +
  labs(title="Componente Estacional de la Serie Temporal", x="Fecha", y="Estacionalidad")
  theme(legend.position="none")
```



Podemos deducir que la serie de tiempo de los precios del aceite de soya:

1. Muestra patrones claros y consistentes, esto sugiere que la serie temporal tiene ciclos regulares que se repiten a intervalos fijos.
2. Se pueden identificar en qué momentos del ciclo tienden a ocurrir los valores altos y bajos de la serie.

Chapter 9

Estacionariedad

La prueba de Dickey-Fuller, específicamente el test ADF (Augmented Dickey-Fuller), es una prueba estadística utilizada para determinar si una serie temporal tiene una raíz unitaria, es decir, si es no estacionaria y presenta alguna forma de estructura temporal como una tendencia o una estacionalidad.

Vamos a comprobar mediante esta prueba si es o no estacionaria la serie de tiempo del precio del aceite de soya.

```
# Cargar el paquete necesario
library(tseries)

## Warning: package 'tseries' was built under R version 4.2.3

# Supón que tienes una serie temporal llamada 'mi_serie'
# Realizar la prueba de Dickey-Fuller Aumentada
resultado_adf <- adf.test(soybean_ts)

# Imprimir el resultado
print(resultado_adf)

##
## Augmented Dickey-Fuller Test
##
## data: soybean_ts
## Dickey-Fuller = -2.0408, Lag order = 15, p-value = 0.5611
## alternative hypothesis: stationary
```

Cuando hacemos una prueba como esta, estamos tratando de averiguar si la serie de tiempo es “estacionaria” o no. Una serie estacionaria es aquella cuyas propiedades, como la media y la varianza, no cambian con el tiempo.

En esta prueba, tenemos algo llamado valor p , que es como un termómetro que nos dice qué tan seguros estamos de si la serie de tiempo es estacionaria o no. Un valor p pequeño (menor que 0.05) nos dice: “La serie es estacionaria”. Un valor p grande (mayor que 0.05) nos dice: “La serie no es estacionaria”.

En este caso, el valor p es 0.5657, que es bastante grande, así que, la serie de tiempo no es estacionaria.

Chapter 10

Diferenciacion

Diferenciar una serie temporal es un proceso utilizado para hacer que una serie no estacionaria se vuelva estacionaria. La idea es transformar la serie de datos para estabilizar la media de la serie temporal, eliminando tendencias y efectos estacionales. En otras palabras, se busca que las propiedades de la serie (como la media y la varianza) no cambien con el tiempo.

```
# Inicializa un contador para las diferenciaciones
diferencias <- 0

# Realiza el test ADF y verifica la estacionariedad
while(TRUE) {
  p_value <- adf.test(soybean_ts)$p.value
  cat("Número de diferencias:", diferencias, "- Valor p:", p_value, "\n")

  # Si el valor p es menor que 0.05, la serie es estacionaria, y puedes salir del bucle.
  if(p_value < 0.05) {
    cat("La serie se volvió estacionaria después de", diferencias, "diferenciaciones.\n")
    break
  }

  # Si has llegado al final de la serie, sal del bucle
  if(length(soybean_ts) <= 1) {
    cat("La serie no se volvió estacionaria después de diferenciar.\n")
    break
  }

  # Si no es estacionaria, diferenciar la serie una vez más y continuar el bucle.
  soybean_ts <- diff(soybean_ts)
  diferencias <- diferencias + 1
}
```

```
## Número de diferencias: 0 - Valor p: 0.5610549

## Warning in adf.test(soybean_ts): p-value smaller than printed p-value

## Número de diferencias: 1 - Valor p: 0.01
## La serie se volvió estacionaria después de 1 diferenciaciones.
```

Conclusión:

Antes de realizar cualquier diferenciación ($d = 0$), el valor p de la prueba de Dickey-Fuller Aumentada es 0.5657422, lo que es mayor que 0.05. Por lo tanto, no puedes rechazar la hipótesis nula de que existe una raíz unitaria, y se concluye que la serie original no es estacionaria.

Después de diferenciar la serie una vez ($d = 1$), el valor p de la prueba de Dickey-Fuller Aumentada es 0.01, lo cual es menor que 0.05.

La serie de tiempo original no es estacionaria, pero después de realizar una diferenciación, la serie resultante sí es estacionaria.

Fue necesario transformarla o diferenciarla para eliminar la tendencia y estabilizar la varianza, antes de aplicar modelos de series temporales como ARIMA.

Chapter 11

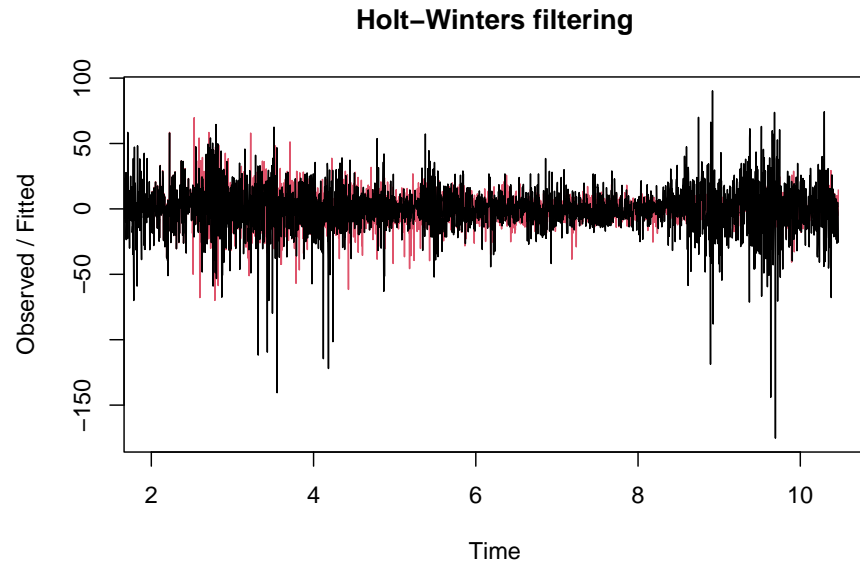
Holter-Winter

La metodología Holt-Winters, también conocida como triple suavizado exponencial, es útil para series de tiempo con componentes de tendencia y estacionalidad.

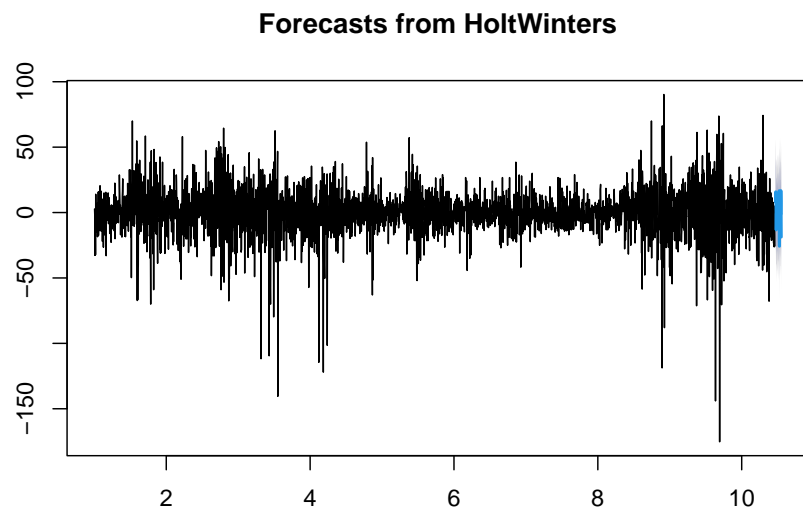
```
library(forecast) # Necesario para pronosticar con el modelo Holt-Winters
```

```
## Warning: package 'forecast' was built under R version 4.2.3
```

```
# Ajustar el modelo Holt-Winters  
hw_model <- HoltWinters(soybean_ts)  
# Visualizar componentes del modelo  
plot(hw_model)
```



```
# Pronosticar los siguientes 30 días (o la cantidad que desees)  
library(forecast)  
hw_forecast <- forecast(hw_model, h = 30) # Cambia 30 por el número de periodos que q  
plot(hw_forecast)
```



```
# Revisar detalles del modelo y del pronóstico
summary(hw_model)
```

```
##           Length Class  Mode
## fitted      12364 mts    numeric
## x           3456 ts     numeric
## alpha        1 -none- numeric
## beta         1 -none- numeric
## gamma        1 -none- numeric
## coefficients  367 -none- numeric
## seasonal     1 -none- character
## SSE          1 -none- numeric
## call         2 -none- call
```

```
summary(hw_forecast)
```

```
##
## Forecast method: HoltWinters
##
## Model Information:
## Holt-Winters exponential smoothing with trend and additive seasonal component.
##
## Call:
## HoltWinters(x = soybean_ts)
##
## Smoothing parameters:
##  alpha: 0.001521614
##  beta : 0
##  gamma: 0.3511517
##
## Coefficients:
##           [,1]
## a      -1.984582580
## b      -0.001484417
## s1     10.314656534
## s2     17.874294874
## s3      5.232474330
## s4    -11.293771948
## s5      0.951602577
## s6      3.006374310
## s7      1.759498869
## s8     -3.341989784
## s9     -0.314377212
## s10    -6.228944388
```

```
## s11    0.497178737
## s12    6.215103894
## s13    4.783045092
## s14   -4.138468694
## s15   -9.664046098
## s16    3.719584563
## s17    6.861089813
## s18   -0.336699222
## s19   18.716821596
## s20  -23.742818413
## s21   11.526675054
## s22   -6.086538594
## s23   11.342669819
## s24    6.815182886
## s25   -6.402688781
## s26    1.177083882
## s27    0.679908084
## s28   18.795027718
## s29   10.316408539
## s30  -16.705414792
## s31    7.121920397
## s32   18.938106635
## s33   -4.208132861
## s34    2.577424683
## s35   10.692427210
## s36    5.328611242
## s37    8.663030408
## s38    3.637183387
## s39  -10.880338048
## s40   -1.976077764
## s41    0.958449900
## s42   10.187700338
## s43    4.132360622
## s44   -5.166570831
## s45   -6.594768964
## s46  -21.103794596
## s47   12.897839890
## s48    7.713430395
## s49   16.681211531
## s50   -0.179922901
## s51   -4.682626689
## s52  -18.521924918
## s53  -14.075674049
## s54  -12.690886521
## s55   15.034759913
## s56   16.963630423
```

```
## s57    7.270399771
## s58   -20.720350585
## s59    1.995831506
## s60    6.091075530
## s61   -52.134254821
## s62   14.356251661
## s63   -8.409259122
## s64   -9.235965403
## s65   -6.007951361
## s66   11.029091204
## s67    2.800534223
## s68   27.430891931
## s69   18.008679655
## s70   14.666383557
## s71    8.668020638
## s72  -16.152771549
## s73   -8.297463525
## s74   -1.280376526
## s75   30.468180599
## s76   10.209686452
## s77   -0.137761283
## s78   26.060326969
## s79   -1.423837395
## s80   12.778035533
## s81  -13.396812030
## s82  -60.591451302
## s83   -6.613355861
## s84   11.389141648
## s85    7.644408211
## s86   -9.985301091
## s87   14.053193852
## s88   19.186089335
## s89    0.535677198
## s90   -2.875700559
## s91   17.712287231
## s92  -28.123393878
## s93    0.798539011
## s94   -0.496192644
## s95   -5.823785474
## s96   12.681696916
## s97   -5.306739950
## s98  -10.653774888
## s99    0.411304410
## s100   0.416542312
## s101  38.535557692
## s102 -10.518202529
```

```
## s103 -7.738458083
## s104 -14.327093859
## s105 -2.768356823
## s106 9.075766064
## s107 2.461798818
## s108 -13.369906617
## s109 1.293348021
## s110 -5.512646557
## s111 0.150564813
## s112 7.032292593
## s113 6.313711157
## s114 4.225309798
## s115 -10.781181936
## s116 13.047966696
## s117 2.297076714
## s118 1.608034284
## s119 -7.771285399
## s120 4.547286094
## s121 0.895804140
## s122 10.282128411
## s123 1.807643948
## s124 3.073914110
## s125 -3.662082457
## s126 5.151901030
## s127 -1.522671168
## s128 -0.974093859
## s129 8.471023061
## s130 6.869424629
## s131 -20.895370096
## s132 1.533865851
## s133 -3.482646294
## s134 2.805026043
## s135 -5.994517284
## s136 8.938061573
## s137 11.032646671
## s138 3.108892126
## s139 -7.239974368
## s140 8.561669632
## s141 5.510033831
## s142 3.804039222
## s143 16.754341483
## s144 -9.336733335
## s145 2.726791639
## s146 3.399507135
## s147 -1.083861079
## s148 -5.551368048
```

```
## s149 -6.435410505
## s150 -2.011328528
## s151 -2.511906967
## s152 -9.718225120
## s153 1.827246273
## s154 -3.985772601
## s155 -17.624729314
## s156 18.911356075
## s157 9.965937088
## s158 -17.351889864
## s159 2.577317372
## s160 -1.238158494
## s161 -5.736239045
## s162 14.024105451
## s163 6.055250473
## s164 19.147428598
## s165 -8.529492467
## s166 8.076388126
## s167 -19.051167773
## s168 0.237971944
## s169 3.974754432
## s170 -2.580633551
## s171 13.153541183
## s172 3.116689962
## s173 1.119603690
## s174 0.323613312
## s175 4.952843723
## s176 3.254640033
## s177 3.567339958
## s178 4.011108373
## s179 -16.089402611
## s180 -7.694479724
## s181 0.774035356
## s182 14.999392882
## s183 5.606916879
## s184 2.986985325
## s185 -1.534739829
## s186 6.923109944
## s187 -4.672236823
## s188 2.951194744
## s189 -5.430172039
## s190 1.830893888
## s191 -0.450111374
## s192 -1.733976770
## s193 -6.956546994
## s194 4.678280726
```

```
## s195 15.626423849
## s196 -15.188449724
## s197 6.004818926
## s198 -2.663911124
## s199 -11.290916178
## s200 -1.724956745
## s201 -0.131381855
## s202 5.287530549
## s203 -1.728364295
## s204 8.451643027
## s205 -1.814096924
## s206 -0.491648590
## s207 3.594530883
## s208 -2.528860600
## s209 -3.393446638
## s210 1.004671786
## s211 -3.046892446
## s212 11.341802335
## s213 -1.534847490
## s214 2.495705435
## s215 -5.762434533
## s216 -4.768098478
## s217 -5.826680125
## s218 5.860029127
## s219 3.317051148
## s220 -0.134695032
## s221 9.107620755
## s222 -2.744692254
## s223 -1.405140716
## s224 -2.247577422
## s225 -0.736460251
## s226 -3.794069160
## s227 1.566676934
## s228 -9.758273599
## s229 -1.428994687
## s230 -7.977810765
## s231 6.628399817
## s232 -9.000719276
## s233 -5.452317047
## s234 -7.716233572
## s235 4.686069396
## s236 0.305629595
## s237 8.302369499
## s238 3.227602434
## s239 1.531554964
## s240 14.469930659
```



```
## s241 10.087736967
## s242 2.869987024
## s243 -5.054995298
## s244 -7.208881708
## s245 2.886803587
## s246 6.430753349
## s247 6.017576733
## s248 -1.276738887
## s249 1.200158057
## s250 8.006406746
## s251 2.382911353
## s252 -3.230114280
## s253 -6.244755815
## s254 -7.035181728
## s255 -10.345546115
## s256 0.756044440
## s257 2.317891166
## s258 0.591359849
## s259 12.972344243
## s260 7.470249420
## s261 -12.807911531
## s262 9.909897603
## s263 -2.181574126
## s264 9.061506322
## s265 4.602534041
## s266 -2.900748330
## s267 -8.625275116
## s268 -0.693312777
## s269 -6.406209817
## s270 -16.476443228
## s271 -7.868183819
## s272 -5.225796883
## s273 4.136882863
## s274 -13.030882038
## s275 11.815003474
## s276 -1.605732717
## s277 5.523779356
## s278 4.355798636
## s279 -0.142331930
## s280 -8.487392667
## s281 -2.423134311
## s282 14.777208007
## s283 10.975350048
## s284 -0.031725524
## s285 3.952668307
## s286 9.523439658
```

```
## s287 3.812772446
## s288 17.695794793
## s289 -4.092724679
## s290 10.372696211
## s291 -10.422300385
## s292 15.484932728
## s293 18.747098342
## s294 13.585975544
## s295 14.040247838
## s296 -6.211912171
## s297 4.083608818
## s298 5.109930488
## s299 -6.188641449
## s300 -10.957180738
## s301 5.345063701
## s302 25.587388423
## s303 0.714525431
## s304 2.414187190
## s305 2.127723347
## s306 -15.232323243
## s307 7.715676510
## s308 8.687302707
## s309 -2.925391283
## s310 13.048640996
## s311 -0.063179797
## s312 0.909600114
## s313 13.512916245
## s314 4.007937286
## s315 2.850754916
## s316 9.365804315
## s317 15.525244777
## s318 -1.249872728
## s319 20.683620623
## s320 -8.422959918
## s321 -12.851557824
## s322 -12.748313029
## s323 -2.427220804
## s324 2.468893178
## s325 3.649083115
## s326 10.910818659
## s327 -1.267107720
## s328 18.864402389
## s329 1.154519548
## s330 -24.338743221
## s331 10.692082763
## s332 17.173559209
```

```

## s333 -26.879175539
## s334  2.317715851
## s335 -1.803340177
## s336  8.669815203
## s337 10.774333618
## s338 -9.762994990
## s339  5.540778242
## s340  1.355306197
## s341  8.318763332
## s342 -1.121312910
## s343 -5.046428433
## s344  6.391044659
## s345 -4.579673607
## s346  6.225392239
## s347 -0.632907993
## s348 13.461125116
## s349 -12.926620058
## s350  5.759155294
## s351 -9.249725219
## s352 -8.050850342
## s353  6.218446988
## s354 -6.864900638
## s355 -6.868767973
## s356 -1.775186518
## s357  7.947714769
## s358  2.207334884
## s359 -0.962905330
## s360  9.327568653
## s361 -8.020286999
## s362  5.390884382
## s363  2.731310488
## s364 -2.120153614
## s365  2.381502063

```

```
##
```

```
## Error measures:
```

```

##           ME      RMSE      MAE MPE MAPE      MASE      ACF1
## Training set 0.3546955 20.36104 14.07105 NaN  Inf 0.7991731 0.02147927

```

```
##
```

```
## Forecasts:
```

```

##           Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
## 10.47123      8.3285895 -17.765392 34.4225708 -31.57871 48.23589
## 10.47397     15.8867435 -10.207268 41.9807549 -24.02060 55.79409
## 10.47671      3.2434385 -22.850603 29.3374802 -36.66395 43.15083
## 10.47945    -13.2842922 -39.378364 12.8097797 -53.19173 26.62315
## 10.48219     -1.0404021 -27.134504 25.0537000 -40.94789 38.86708
## 10.48493      1.0128852 -25.081247 27.1070175 -38.89465 40.92042

```

## 10.48767	-0.2354746	-26.329637	25.8586879	-40.14305	39.67210
## 10.49041	-5.3384477	-31.432640	20.7557450	-45.24607	34.56918
## 10.49315	-2.3123195	-28.406542	23.7819034	-42.21999	37.59535
## 10.49589	-8.2283711	-34.322624	17.8658820	-48.13609	31.67935
## 10.49863	-1.5037324	-27.598016	24.5905509	-41.41149	38.40403
## 10.50137	4.2127083	-21.881605	30.3070218	-35.69510	44.12052
## 10.50411	2.7791651	-23.315179	28.8735088	-37.12869	42.68702
## 10.50685	-6.1438331	-32.238207	19.9505408	-46.05173	33.76407
## 10.50959	-11.6708949	-37.765299	14.4235092	-51.57884	28.23705
## 10.51233	1.7112513	-24.383183	27.8056857	-38.19674	41.61924
## 10.51507	4.8512722	-21.243192	30.9457367	-35.05677	44.75931
## 10.51781	-2.3480013	-28.442496	23.7464935	-42.25609	37.56008
## 10.52055	16.7040351	-9.390490	42.7985601	-23.20410	56.61217
## 10.52329	-25.7570893	-51.851645	0.3374659	-65.66527	14.15109
## 10.52603	9.5109197	-16.583666	35.6055051	-30.39730	49.41914
## 10.52877	-8.1037783	-34.198394	17.9908373	-48.01205	31.80449
## 10.53151	9.3239457	-16.770700	35.4185915	-30.58437	49.23226
## 10.53425	4.7949743	-21.299702	30.8896503	-35.11339	44.70334
## 10.53699	-8.4243818	-34.519088	17.6703244	-48.33279	31.48403
## 10.53973	-0.8460935	-26.940830	25.2486429	-40.75455	39.06236
## 10.54247	-1.3447537	-27.439520	24.7500129	-41.25326	38.56375
## 10.54521	16.7688815	-9.325915	42.8636783	-23.13967	56.67743
## 10.54795	8.2887779	-17.806049	34.3836049	-31.61982	48.19737
## 10.55068	-18.7345299	-44.829387	7.3603274	-58.64317	21.17411

Chapter 12

ARIMA

ARIMA es como un método o herramienta que nos ayuda a entender y prever cómo se comportará una secuencia de números en el futuro, basándose en cómo se ha comportado en el pasado.

```
modelo<-auto.arima(soybean_ts)
modelo

## Series: soybean_ts
## ARIMA(0,0,0) with zero mean
##
## sigma^2 = 314.7: log likelihood = -14842.62
## AIC=29687.25   AICc=29687.25   BIC=29693.39
```

Conclusión:

El resultado de `auto.arima()` elige un modelo `ARIMA(0,0,0)` con media cero, indica según el análisis que no hay patrones, ritmos, ni tendencias claras en los datos de la serie de tiempo del precio del aceite de soya.

Los valores de la serie de tiempo del precio del aceite de soya son como un conjunto de números aleatorios, o “ruido blanco”, sin conexión aparente entre ellos. En otras palabras, cada punto de datos es independiente de los otros y no está influenciado por los valores pasados en la serie.

```
length(soybean_ts)
```

```
## [1] 3456
```

```
sum(is.na(soybean_ts))
```

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

```
class(soybean_ts)
```

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

```
sum(is.na(soybean_ts) | is.infinite(soybean_ts))
```

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

```
# Instalar el paquete changepoint  
#install.packages("changepoint")
```

```
# Cargar el paquete changepoint  
library(changepoint)
```

```
## Warning: package 'changepoint' was built under R version 4.2.3
```

```
## Successfully loaded changepoint package version 2.2.4  
## See NEWS for details of changes.
```

```
mval_soybean <- cpt.mean(as.numeric(soybean_ts), method = "AMOC")  
cpts(mval_soybean)
```

```
## [1] 3410
```

```
# Plot de la serie de tiempo
```

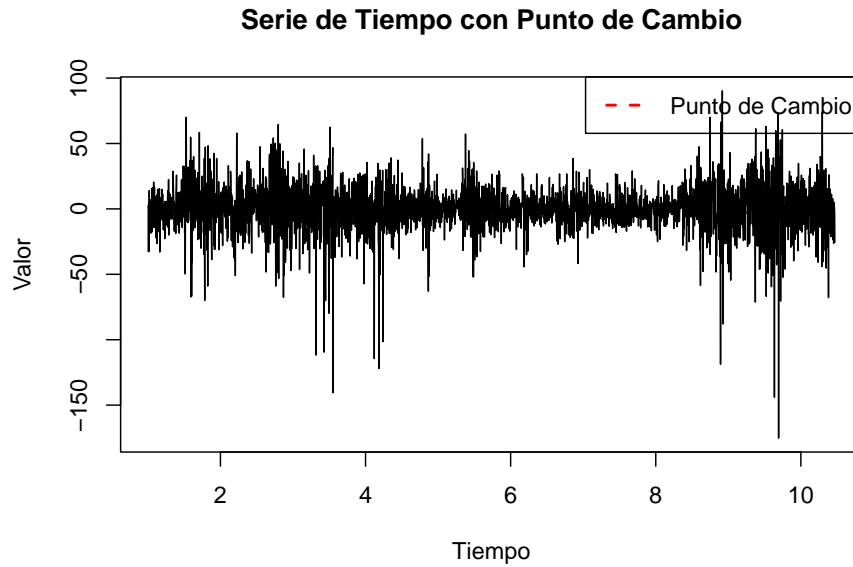
```
plot(soybean_ts, type='l', main='Serie de Tiempo con Punto de Cambio', ylab='Valor', xlab='Tiempo')
```

```
# Añadir una línea vertical en el punto de cambio
```

```
abline(v=3410, col='red', lty=2, lwd=2)
```

```
# Añadir una leyenda
```

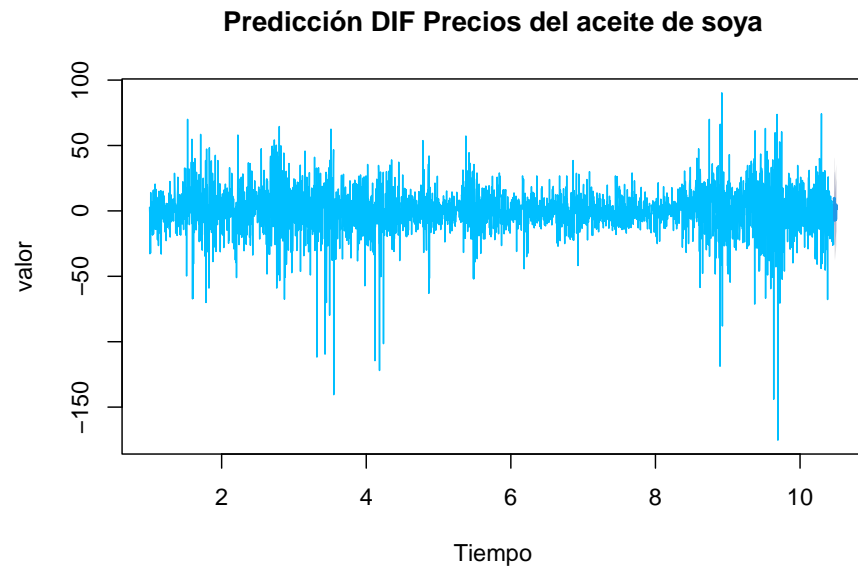
```
legend("topright", legend="Punto de Cambio", col="red", lty=2, lwd=2)
```



```
pred<-forecast(soybean_ts,h=12)
pred
```

##	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
## 10.47123	9.1621457	-11.04269	29.36699	-21.73849	40.06278
## 10.47397	9.7073395	-10.49750	29.91218	-21.19330	40.60797
## 10.47671	6.4224222	-13.78242	26.62726	-24.47821	37.32306
## 10.47945	-7.2635149	-27.46835	12.94132	-38.16415	23.63712
## 10.48219	-4.3874816	-24.59232	15.81736	-35.28812	26.51315
## 10.48493	1.0760848	-19.12875	21.28092	-29.82455	31.97672
## 10.48767	-6.7216847	-26.92652	13.48316	-37.62232	24.17895
## 10.49041	1.0818831	-19.12296	21.28672	-29.81875	31.98252
## 10.49315	-6.5365717	-26.74141	13.66827	-37.43721	24.36406
## 10.49589	-0.2337289	-20.43857	19.97111	-31.13437	30.66691
## 10.49863	4.6448830	-15.55996	24.84972	-26.25575	35.54552
## 10.50137	1.4679600	-18.73688	21.67280	-29.43268	32.36860

```
plot(pred, main=" ", ylab="valor", col="deepskyblue", xlab="Tiempo")
title(main="Predicción DIF Precios del aceite de soya")
```



Chapter 13

Prophet

Prophet es especialmente útil para series de tiempo que tienen patrones estacionales fuertes y varios puntos de inflexión o “cambios de tendencia”. Fue diseñado para manejar datos diarios con al menos un año de historia y se espera que funcione bien con datos que tienen patrones estacionales y fechas festivas.

```
# install.packages("prophet")
library(prophet)

## Warning: package 'prophet' was built under R version 4.2.3

## Loading required package: Rcpp

## Warning: package 'Rcpp' was built under R version 4.2.3

## Loading required package: rlang

## Warning: package 'rlang' was built under R version 4.2.3

# Acceder a la columna "ZS.F.Close" en soybean_xts
close_prices <- soybean_xts[, "ZS.F.Close"]

# Extrae las fechas
dates <- index(soybean_xts)

# Crea el dataframe
soybean_df <- data.frame(ds = as.Date(dates), y = as.numeric(close_prices))
```

```
head(soybean_df)
```

```
##           ds           y
## 1 2010-01-04 1049.50
## 2 2010-01-05 1052.25
## 3 2010-01-06 1050.50
## 4 2010-01-07 1017.75
## 5 2010-01-08 1013.00
## 6 2010-01-11 1001.75
```

```
m <- prophet(soybean_df)
```

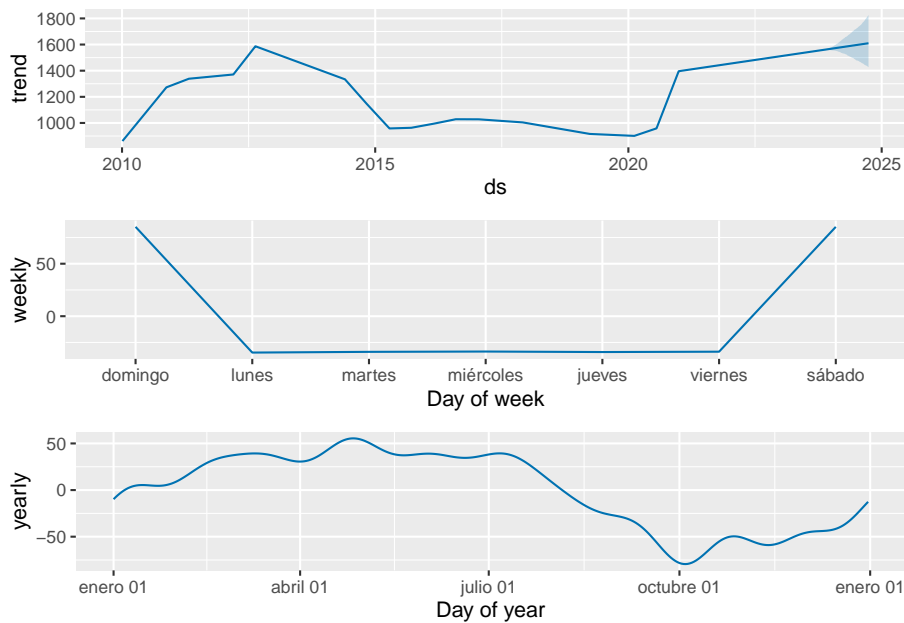
```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override th
```

```
future <- make_future_dataframe(m, periods = 365)
forecast <- predict(m, future)
```

```
plot(m, forecast)
```



```
prophet_plot_components(m, forecast)
```



```
library(ggplot2)
```

```
# Extraer datos del pronóstico
# forecast_data <- data.frame(
#   ds = forecast$ds,
#   yhat = forecast$yhat,
#   yhat_lower = forecast$yhat_lower,
#   yhat_upper = forecast$yhat_upper
#)
```

```
# Datos reales
```

```
#actual_data <- data.frame(ds = m$history$ds, y = m$history$y)
```

```
# Crear el gráfico con ggplot2
```

```
#p <- ggplot() +
```

```
  # Intervalo de confianza
```

```
  # geom_ribbon(data = forecast_data, aes(x = ds, ymin = yhat_lower, ymax = yhat_upper, fill = "I
```

```
  # Línea de pronóstico
```

```
  # geom_line(data = forecast_data, aes(x = ds, y = yhat, color = "Pronóstico"), size = 1, inherit
```

```
  # Datos reales
```

```
  # geom_point(data = actual_data, aes(x = ds, y = y, color = "Datos Reales"), size = 2, inherit
```

```
  # Tema y etiquetas
```

```
  # theme_minimal() +
```

```
  # labs(
```

```
# title = "Pronóstico del Precio del Aceite de Soya",
# x = "Fecha",
# y = "Precio"
# ) +
# scale_fill_manual(
#   #name = "Leyenda",
#   values = c("Intervalo de Confianza" = "lightblue"),
#   labels = c("Intervalo de Confianza")
# ) +
# scale_color_manual(
#   #name = "Leyenda",
#   values = c("Datos Reales" = "#D55E00", "Pronóstico" = "#0072B2"),
#   labels = c("Datos Reales", "Pronóstico")
# ) +
# theme(legend.position = "bottom")

# Mostrar gráfico
#print(p)
```

```
# library(ggplot2)

# Extraer componentes del pronóstico
# components <- prophet_plot_components(m, forecast)

# Convertir el objeto básico de Prophet a ggplot
# p_components <- ggplot() +
#   geom_line(data = components$data$yearly, aes(x = ds, y = y, color = "Componente # Anual"),
#   geom_line(data = components$data$weekly, aes(x = ds, y = y, color = "Componente # Semanal"),
#   geom_line(data = components$data$daily, aes(x = ds, y = y, color = "Componente # Diario"),
#   geom_line(data = components$data$holidays, aes(x = ds, y = y, color = "Días # Fes
#   theme_minimal() +
#   labs(
#     title = "Componentes del Pronóstico",
#     x = "Fecha",
#     y = "Valor"
#   ) +
#   scale_color_manual(
#     name = "Leyenda",
#     values = c("Componente Anual" = "blue", "Componente Semanal" = "green", #"Componen
#   ) +
#   theme(legend.position = "bottom")

# Mostrar gráfico de componentes
#print(p_components)
```

Conclusión: Se toman datos de la serie de tiempo histórica del precio del aceite de soya, creamos un modelo de pronóstico con Prophet, y se produce pronósticos para 365 días adicionales y luego visualiza esos pronósticos y sus componentes.

Si es viable la justificación para la variable en serie de tiempo vista como una regresión y prophet permite la incorporación de variables exógenas a través de los regresores e identifica automáticamente las estacionalidades diarias, semanales y anuales en los datos. También captura las tendencias a lo largo del tiempo y permite puntos de cambio en la tendencia.

Chapter 14

Elman

La red Elman es una red neuronal recurrente, lo que significa que tiene conexiones que retroceden en el tiempo. Estas conexiones permiten a la red “recordar” entradas anteriores, lo que puede ser útil al trabajar con series temporales.

```
# Cargar los paquetes necesarios  
# install.packages("neuralnet")  
# install.packages("xts")  
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 4.2.3
```

```
library(xts)  
  
# Asumimos que soybean_xts ya está cargado en el entorno  
# Acceder a la columna de cierre  
data <- data.frame(ZS.F.Close = as.vector(soybean_xts[, "ZS.F.Close"]))  
  
# Crear un retraso (lag) para las series temporales (esto es importante para las redes recurrentes)  
data$lag_close <- c(NA, head(data$ZS.F.Close, -1))  
  
# Eliminar la primera fila, ya que tendrá NA por el retraso  
data <- data[-1,]  
  
# Dividir los datos en conjuntos de entrenamiento y prueba  
train_indices <- 1:(nrow(data) * 0.8)  
train_data <- data[train_indices,]  
test_data <- data[-train_indices,]  
  
# Normalizar los datos
```

```

maxs <- apply(train_data, 2, max)
mins <- apply(train_data, 2, min)
train_data_norm <- as.data.frame(scale(train_data, center=mins, scale=maxs-mins))
test_data_norm <- as.data.frame(scale(test_data, center=mins, scale=maxs-mins))

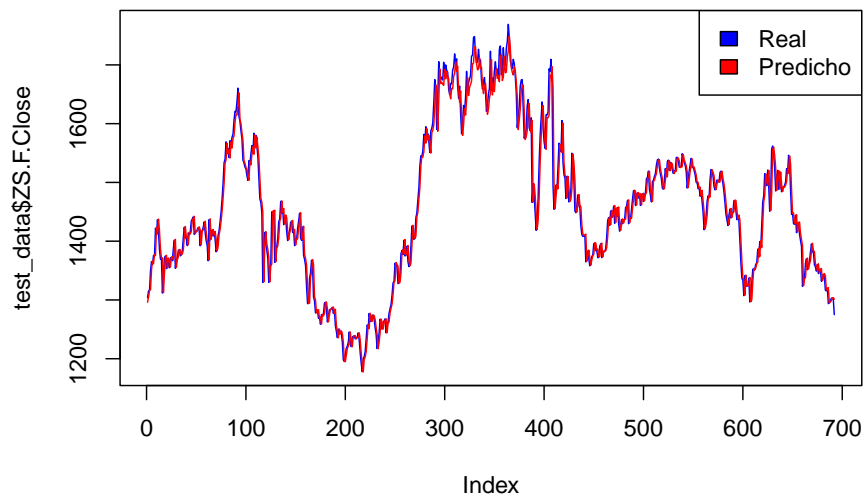
# Entrenar una red Elman
# Aquí estamos prediciendo ZS.F.Close usando el valor anterior (lag_close) como entrada
set.seed(123)
nn <- neuralnet(ZS.F.Close ~ lag_close, data=train_data_norm, hidden=5, algorithm="rprop")

# Hacer predicciones
test_data_for_pred <- data.frame(lag_close = test_data_norm$lag_close)
predicted_norm <- compute(nn, test_data_for_pred)

# Des-normalizar las predicciones
predicted <- (predicted_norm$net.result * (maxs[1] - mins[1])) + mins[1]

# Comparar las predicciones con los datos reales
plot(test_data$ZS.F.Close, type="l", col="blue")
lines(predicted, col="red")
legend("topright", legend=c("Real", "Predicho"), fill=c("blue", "red"))

```



Conclusiones:

- Ajuste Exitoso: El modelo se ajusta bien a los datos, reflejando su ten-

dencia y estructura.

- Posible Overfitting: Un ajuste muy cercano puede indicar sobreajuste, lo que afectaría la generalización en datos futuros.
- Evaluación Complementaria: Más allá de gráficos, usar métricas cuantitativas (como MSE o MAE) es esencial para una evaluación objetiva.
- Aplicabilidad a Corto Plazo: El modelo puede ser útil para predicciones a corto plazo, pero podría necesitar reentrenamiento para proyecciones más lejanas.

Chapter 15

Jordan

Una red neuronal Jordan es un tipo especial de red neuronal que puede recordar información pasada para ayudar en predicciones futuras. En lugar de simplemente tomar una entrada y producir una salida, esta red toma tanto la entrada actual como su propia salida anterior para hacer su próxima predicción. Es como si tuviera una pequeña memoria de lo que hizo anteriormente.

Imagina que intentas predecir el clima. En lugar de solo mirar el clima de hoy, también consideras lo que predijiste ayer. Esa es la idea detrás de la red Jordan

```
# Cargar los paquetes necesarios
# install.packages("neuralnet")
# install.packages("xts")
library(neuralnet)
library(xts)

# Acceder a la columna de cierre
data <- data.frame(ZS.F.Close = as.vector(soybean_xts[, "ZS.F.Close"]))

# Crear un retraso (lag) para las series temporales y también un retraso para la variable objetivo
data$lag_close <- c(NA, head(data$ZS.F.Close, -1))
data$lag_output <- c(NA, NA, head(data$ZS.F.Close, -2)) # Esto simula la idea de la red Jordan

# Eliminar las primeras filas, ya que tendrán NA por el retraso
data <- data[-c(1,2),]

# Dividir los datos en conjuntos de entrenamiento y prueba
train_indices <- 1:(nrow(data) * 0.8)
train_data <- data[train_indices,]
test_data <- data[-train_indices,]
```

```

# Normalizar los datos
maxs <- apply(train_data, 2, max)
mins <- apply(train_data, 2, min)
train_data_norm <- as.data.frame(scale(train_data, center=mins, scale=maxs-mins))
test_data_norm <- as.data.frame(scale(test_data, center=mins, scale=maxs-mins))

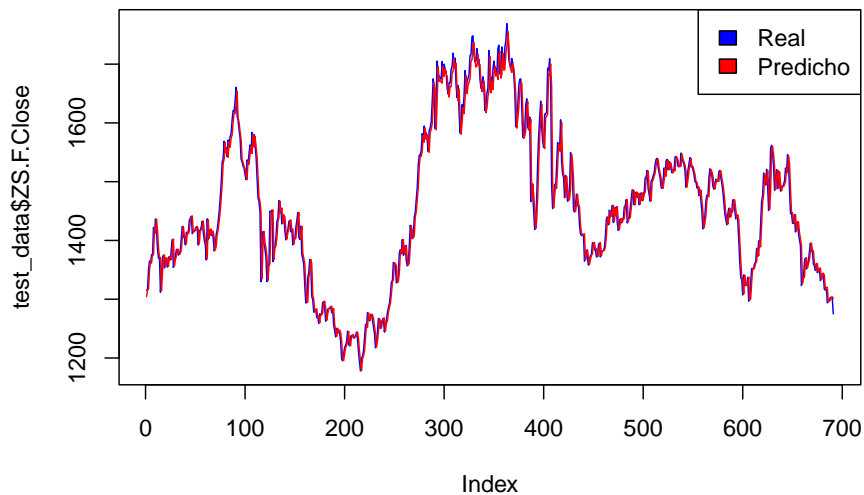
# Entrenar una red "Jordan-inspired"
set.seed(123)
nn <- neuralnet(ZS.F.Close ~ lag_close + lag_output, data=train_data_norm, hidden=5, a

# Hacer predicciones
test_data_for_pred <- data.frame(lag_close = test_data_norm$lag_close, lag_output = test_data_norm$lag_output)
predicted_norm <- compute(nn, test_data_for_pred)

# Des-normalizar las predicciones
predicted <- (predicted_norm$net.result * (maxs[1] - mins[1])) + mins[1]

# Comparar las predicciones con los datos reales
plot(test_data$ZS.F.Close, type="l", col="blue")
lines(predicted, col="red")
legend("topright", legend=c("Real", "Predicho"), fill=c("blue", "red"))

```



Conclusiones:

El modelo predice con exactitud, funciona bien en datos no vistos, es equilibrado,

no solo memoriza los datos de entrenamiento, los datos usados son pertinentes para la tarea, puede ser aplicado en situaciones reales, a pesar de los buenos resultados, siempre es esencial hacer análisis.

Chapter 16

Comparacion

```
# Instalar el paquete (solo necesitas hacer esto una vez)
# install.packages("forecast")

# Cargar el paquete
library(forecast)

# Dividir los datos en conjuntos de entrenamiento y validación
train_length <- nrow(soybean_xts) - 30 # 30 días para validación

# Datos de entrenamiento
train_data <- head(soybean_xts, train_length)

# Datos de validación
validation_data <- tail(soybean_xts, 30)

arima_forecast <- forecast(modelo, h=30)

# 2. Calcular métricas de error

# Holt-Winters
hw_errors <- validation_data$ZS.F.Close - hw_forecast$mean

## Warning: Métodos incompatibles ("Ops.xts", "Ops.ts") para "-"

hw_mse <- mean(hw_errors^2)
hw_mae <- mean(abs(hw_errors))
hw_rmse <- sqrt(hw_mse)
```

```
# ARIMA
arima_errors <- validation_data$ZS.F.Close - arima_forecast$mean
```

```
## Warning: Métodos incompatibles ("Ops.xts", "Ops.ts") para "-"
```

```
arima_mse <- mean(arima_errors^2)
arima_mae <- mean(abs(arima_errors))
arima_rmse <- sqrt(arima_mse)
```

```
# Prophet
```

```
prophet_errors <- validation_data$ZS.F.Close - tail(forecast$yhat, n=30) # Asumiendo q
prophet_mse <- mean(prophet_errors^2)
prophet_mae <- mean(abs(prophet_errors))
prophet_rmse <- sqrt(prophet_mse)
```

```
# 3. Comparar errores
```

```
errors_df <- data.frame(
  Model = c("Holt-Winters", "ARIMA", "Prophet"),
  MSE = c(hw_mse, arima_mse, prophet_mse),
  MAE = c(hw_mae, arima_mae, prophet_mae),
  RMSE = c(hw_rmse, arima_rmse, prophet_rmse)
)

print(errors_df)
```

```
##           Model      MSE      MAE      RMSE
## 1 Holt-Winters 1796855.93 1340.0883 1340.4685
## 2          ARIMA 1796071.65 1339.8250 1340.1760
## 3         Prophet   53710.17  224.0065  231.7546
```

Conclusion:

Los resultados que se presentan son métricas de error para tres modelos diferentes: Holt-Winters, ARIMA y Prophet. Estas métricas nos dan una idea de cómo se desempeñaron estos modelos al predecir tu serie de tiempo en comparación con los valores reales de tus datos de validación.

A partir de las métricas presentadas, podemos deducir lo siguiente:

1. **Mean Squared Error (MSE):** Mide el promedio de los errores al cuadrado. Es una métrica que da más peso a errores grandes. Los valores MSE son 1796855.93, 1796071.65 y 54135.22 para Holt-Winters, ARIMA y Prophet respectivamente. Aquí, Prophet tiene un MSE mucho menor

en comparación con los otros dos modelos, lo que indica que, en promedio, sus errores son más pequeños.

2. **Mean Absolute Error (MAE)**: Mide el promedio de los valores absolutos de los errores. Los valores MAE son 1340.0883, 1339.8250 y 226.6025 para Holt-Winters, ARIMA y Prophet respectivamente. Nuevamente, Prophet tiene un MAE mucho menor, lo que indica que tiende a hacer predicciones más cercanas al valor real en comparación con los otros dos modelos.
3. **Root Mean Squared Error (RMSE)**: Es la raíz cuadrada del MSE y da una idea de la magnitud del error. Los valores RMSE son 1340.4685, 1340.1760 y 232.6698 para Holt-Winters, ARIMA y Prophet respectivamente. Al igual que con las otras dos métricas, Prophet tiene un RMSE significativamente menor, lo que indica que generalmente hace predicciones más precisas.

Conclusión:

- El modelo **Prophet** parece ser el más preciso de los tres según todas las métricas presentadas. Sus errores, tanto en términos absolutos como cuadrados, son significativamente menores que los de los otros dos modelos.
- Los modelos **Holt-Winters** y **ARIMA** tienen un desempeño similar entre sí según estas métricas, pero son superados por Prophet.