PRECIOS DEL ORO Y LA PLATA

By: Ael-Dev

Diccionario de variables:

```
"Date": Fecha en la que se registró el precio
"gold": Precio del oro en la fecha dada
"silver": Precio de la plata en la fecha dada
"plat": Precio del platino en la fecha dada
"pall": Precio del paladio en la fecha dada
```

Leer dataset

```
library(readr)

data <- read_csv('https://raw.githubusercontent.com/ecabestadistica/series-temporales-multivariantes/master/2.%20Casos%20de%20estudio%20en%20R/D.%20Comodities%20Oro%20y%20Plata/comodity_price.csv',
                 col_types = cols(
                  #                date = col_double(),
                                  gold = col_double(),
                                  silver = col_double(),
                                  plat = col_double(),
                                  pall = col_double()
                                  )
                 )
class(data)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```
head(data)
```

Date <date>	gold <dbl>	silver <dbl>	plat <dbl>	pall <dbl>
1993-11-01	369.25	4.25	367.2	128.75
1993-12-01	376.30	4.62	374.0	124.00
1994-01-01	394.00	5.24	399.0	126.25
1994-02-01	384.50	5.31	393.0	128.25
1994-03-01	378.75	5.32	392.5	136.10
1994-04-01	391.00	5.78	423.0	135.50

6 rows

Graficando las series

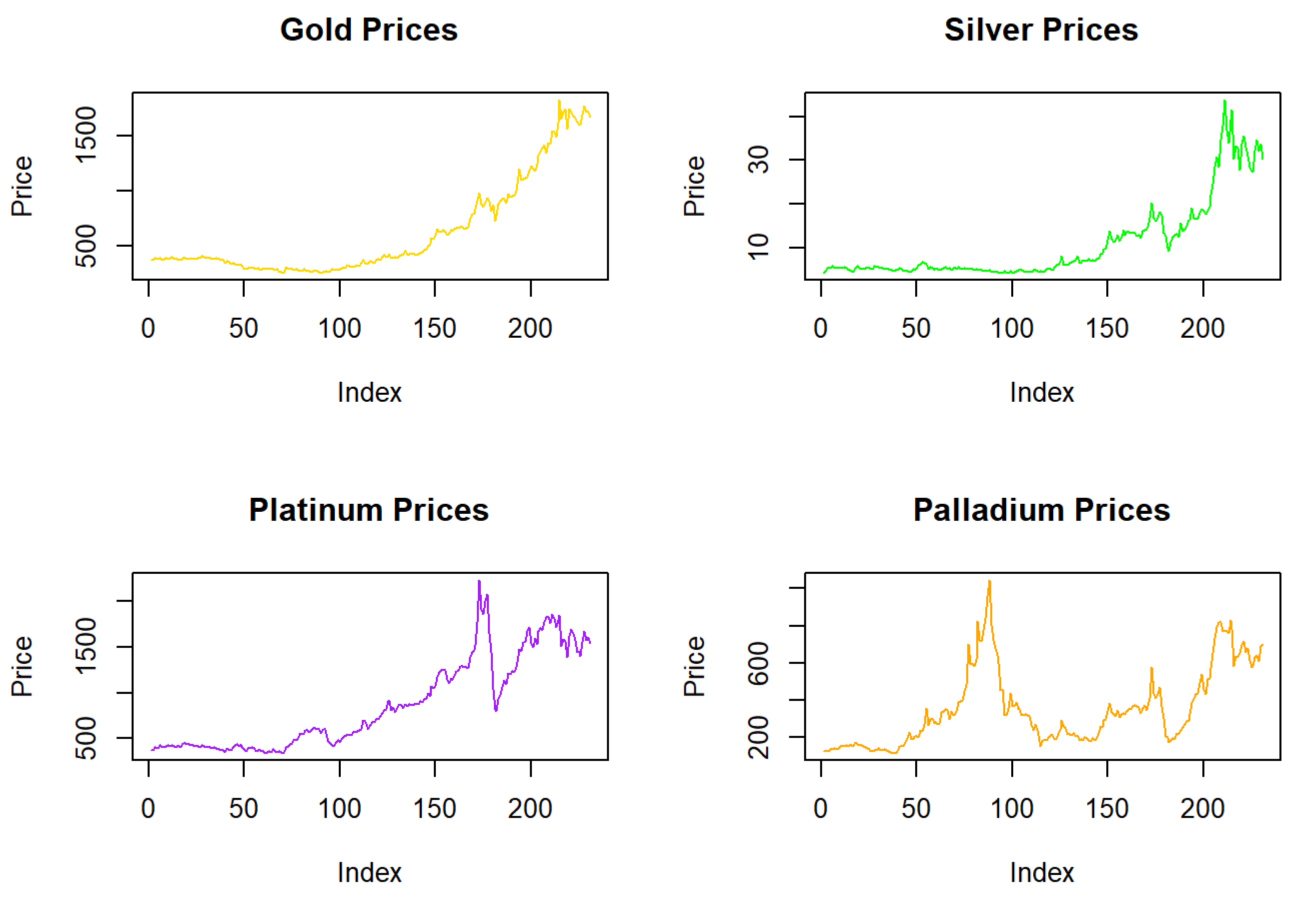
```
# Crear un lienzo para los 4 gráficos
par(mfrow = c(2, 2))

# Gráfico 1: Gold
plot(data$gold, main = "Gold Prices", ylab = "Price", type = "l", col = "gold")

# Gráfico 2: Silver
plot(data$silver, main = "Silver Prices", ylab = "Price", type = "l", col = "green")

# Gráfico 3: Platinum
plot(data$plat, main = "Platinum Prices", ylab = "Price", type = "l", col = "purple")

# Gráfico 4: Palladium
plot(data$pall, main = "Palladium Prices", ylab = "Price", type = "l", col = "orange")
```

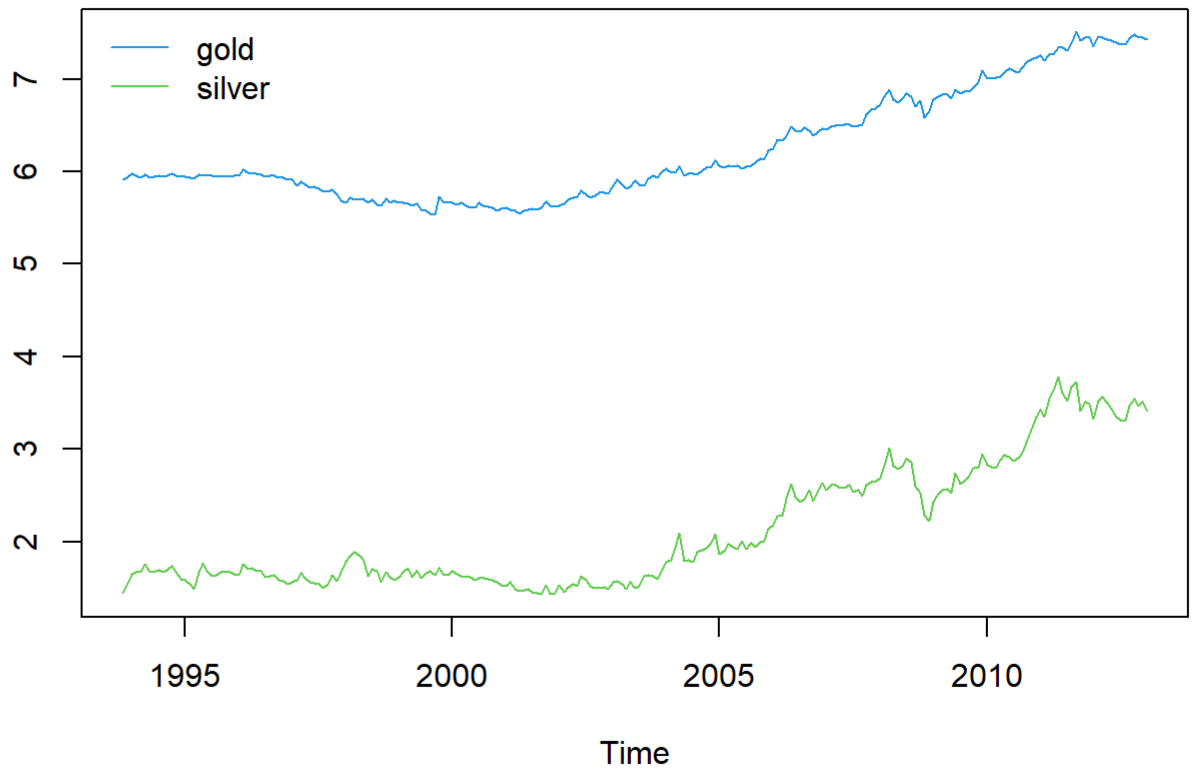


Crear las series de tipo ts

```
# aplicando transformacion logaritmica
gold <- ts(log(data$gold),start=c(1993,11),frequency=12)
silver <- ts(log(data$silver),start=c(1993,11),frequency=12)
plat <- ts(log(data$plat),start=c(1993,11),frequency=12)
pall <- ts(log(data$pall),start=c(1993,11),frequency=12)
```

Graficar Gold y Silver

```
par(mfrow=c(1,1))
plot.ts(cbind(gold,silver), plot.type="single", ylab="",col = 4:3)
legend("topleft",legend=c("gold","silver"),col=4:3,lty=1,bty='n')
```



Una sola serie con las dos

```
data <- ts.union(gold,silver)
```

Prueba estacionariedad

H0: No estacionaria H1: Estacionaria

```
library(tseries)
```

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

```
apply(data, 2, adf.test) #2 para especificar que lo queremos aplicar por columnas
```

```
## $gold
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -1.4954, Lag order = 6, p-value = 0.7879
## alternative hypothesis: stationary
##
##
## $silver
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -1.8285, Lag order = 6, p-value = 0.648
## alternative hypothesis: stationary
```

Los resultados indican que las series no son estacionarias.

Diferenciando las series

```
library(MTS)
```

```
## Warning: package 'MTS' was built under R version 4.3.3
```

```
stnry = diffM(data)
```

Volviendo a hacer el test:

```
apply(stnry, 2, adf.test)
```

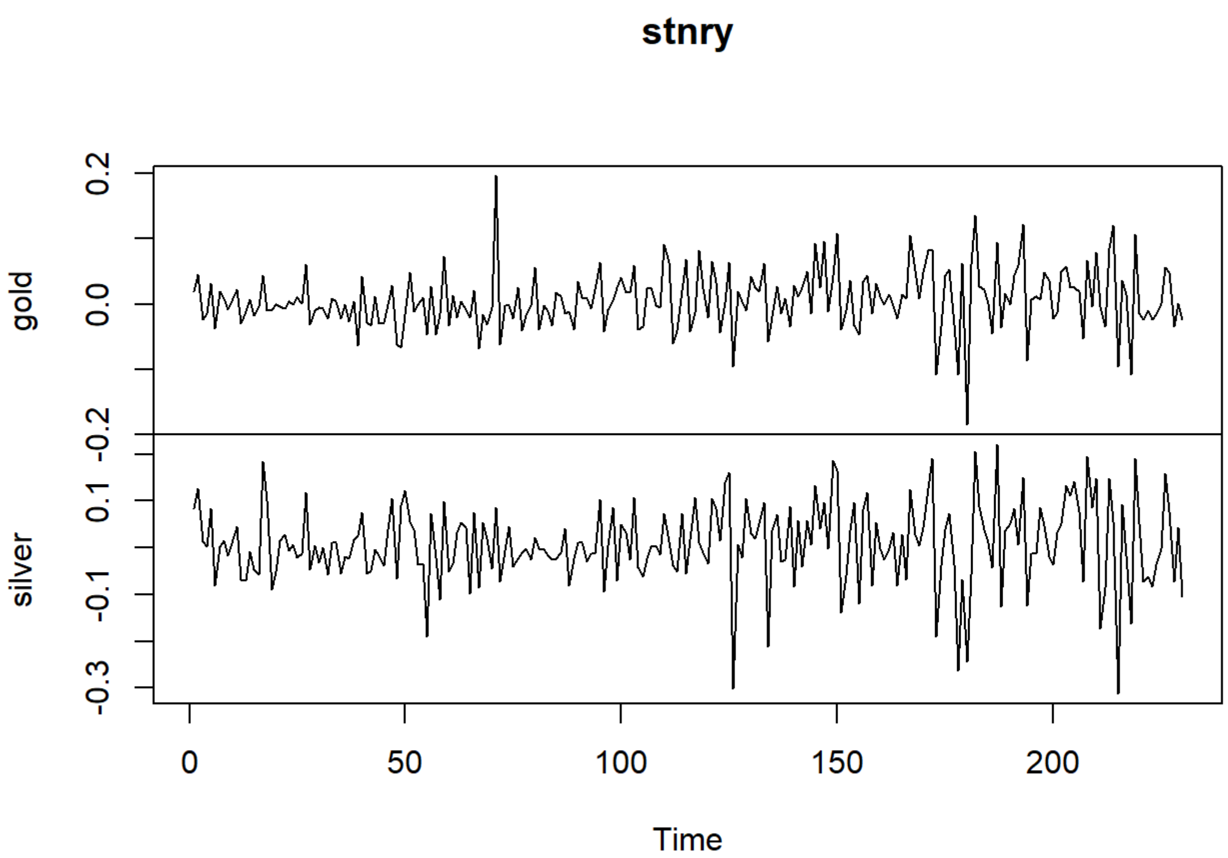
```
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
```

```
## $gold
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -5.8442, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $silver
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -5.8225, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

Las series son estacionarias ($p < 0.05$)

```
plot.ts(stnry)
```



Identificación del orden del modelo

```
library(vars)
```

```
## Warning: package 'vars' was built under R version 4.3.3
```

```
## Loading required package: MASS
```

```
## Loading required package: strucchange
```

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

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

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

```
## Loading required package: sandwich
```

```
## Loading required package: urca
```

```
## Loading required package: lmtest
```

```
##
## Attaching package: 'vars'
```

```
## The following object is masked from 'package:MTS':
##
##     VAR
```

```
VARselect(stnry, type = "none", lag.max = 10)
```

```
## $selection
## AIC(n)  HQ(n)  SC(n) FPE(n)
##      1      1      1      1
##
## $criteria
##                    1             2             3             4             5
## AIC(n) -1.167475e+01 -1.166544e+01 -1.165348e+01 -1.162257e+01 -1.161114e+01
## HQ(n)  -1.164983e+01 -1.161560e+01 -1.157873e+01 -1.152290e+01 -1.148656e+01
## SC(n)  -1.161305e+01 -1.154203e+01 -1.146838e+01 -1.137576e+01 -1.130263e+01
## FPE(n)  8.505922e-06  8.585559e-06  8.688974e-06  8.962110e-06  9.065638e-06
##                    6             7             8             9            10
## AIC(n) -1.158784e+01 -1.156195e+01 -1.152847e+01 -1.154591e+01 -1.151690e+01
## HQ(n)  -1.143834e+01 -1.138753e+01 -1.132913e+01 -1.132165e+01 -1.126773e+01
## SC(n)  -1.121763e+01 -1.113004e+01 -1.103485e+01 -1.099059e+01 -1.089988e+01
## FPE(n)  9.280242e-06  9.524828e-06  9.850826e-06  9.682651e-06  9.970329e-06
```

Creando el modelo

```
var.a <- vars::VAR(stnry,

                   lag.max = 10,

                   ic = "AIC",

                   type = "none")

summary(var.a)
```

```
##
## VAR Estimation Results:
## =========================
## Endogenous variables: gold, silver
## Deterministic variables: none
## Sample size: 229
## Log Likelihood: 697.534
## Roots of the characteristic polynomial:
## 0.1226 0.01734
## Call:
## vars::VAR(y = stnry, type = "none", lag.max = 10, ic = "AIC")
##
##
## Estimation results for equation gold:
## =====================================
## gold = gold.l1 + silver.l1
##
##           Estimate Std. Error t value Pr(>|t|)
## gold.l1   -0.14190    0.09417  -1.507    0.133
## silver.l1  0.01086    0.05094   0.213    0.831
##
##
## Residual standard error: 0.04643 on 227 degrees of freedom
## Multiple R-Squared: 0.01646, Adjusted R-squared: 0.007792
## F-statistic: 1.899 on 2 and 227 DF,  p-value: 0.1521
##
##
## Estimation results for equation silver:
## =======================================
## silver = gold.l1 + silver.l1
##
##           Estimate Std. Error t value Pr(>|t|)
## gold.l1   -0.28304    0.17425  -1.624    0.106
## silver.l1  0.03664    0.09425   0.389    0.698
##
##
## Residual standard error: 0.08592 on 227 degrees of freedom
## Multiple R-Squared: 0.01672, Adjusted R-squared: 0.008055
## F-statistic:  1.93 on 2 and 227 DF,  p-value: 0.1475
##
##
##
## Covariance matrix of residuals:
##            gold   silver
## gold   0.002101 0.002761
## silver 0.002761 0.007286
##
## Correlation matrix of residuals:
##          gold silver
## gold   1.0000 0.7056
## silver 0.7056 1.0000
```

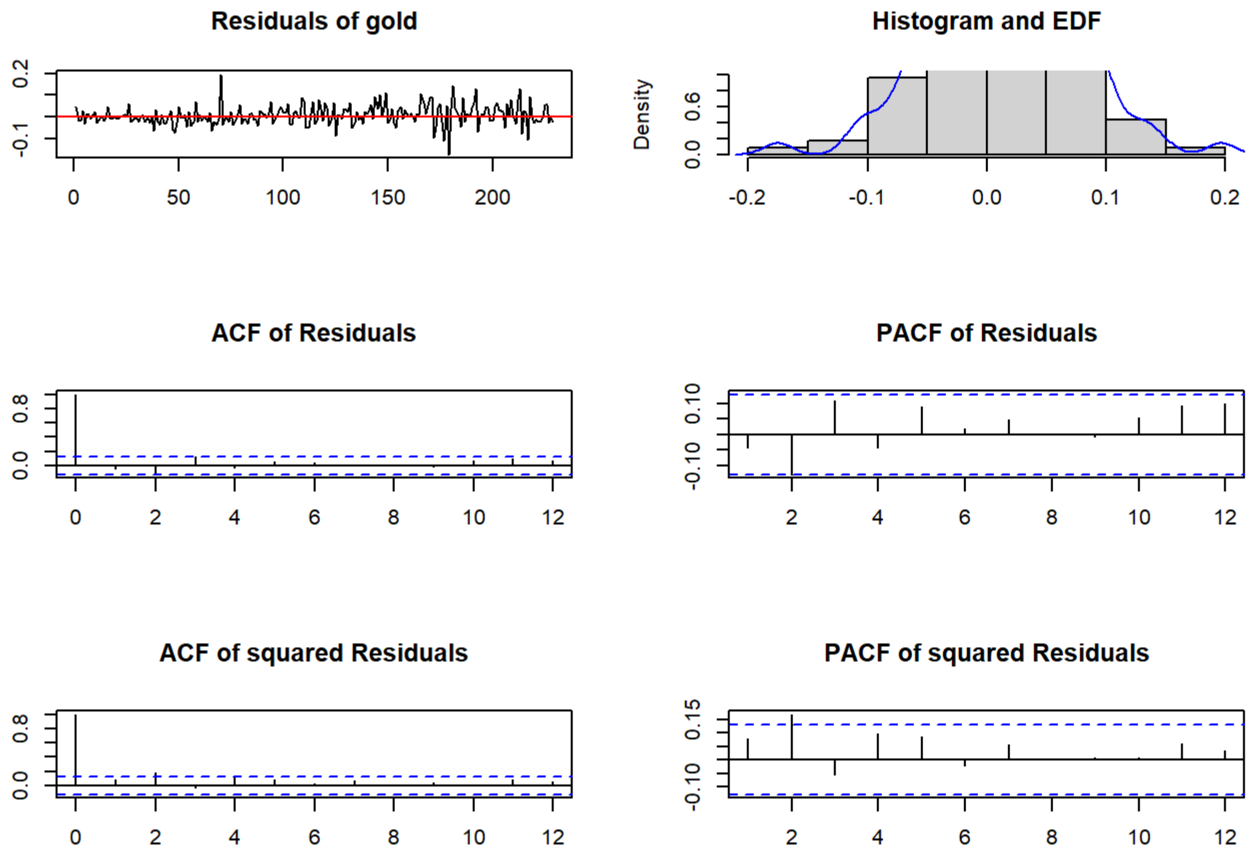
Diagnosis del modelo (Portmanteau test para objetos var)

h0: no hay autocorrelación en los residuos de la serie temporal h1: existen autocorrelaciones significativas en los residuos de la serie temporal

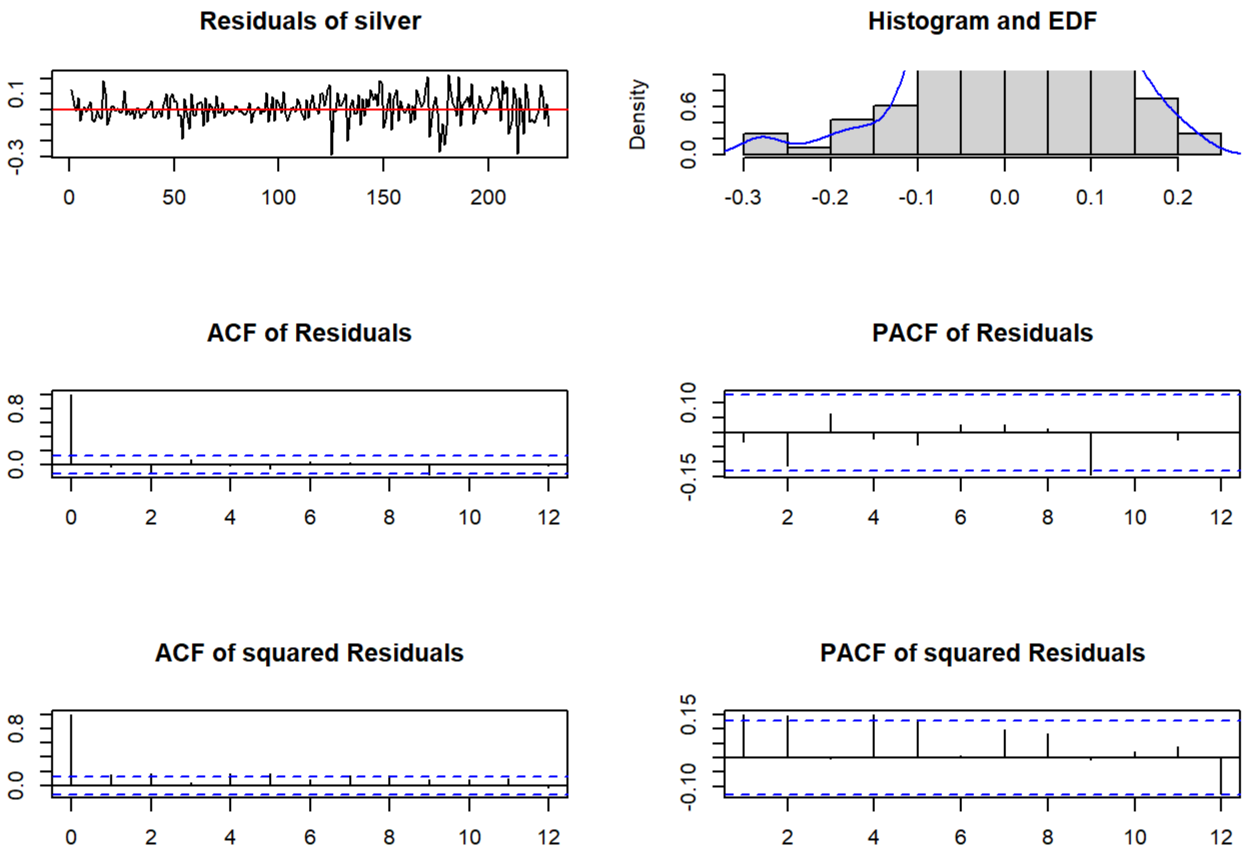
```
bv.serial=serial.test(var.a)
bv.serial
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var.a
## Chi-squared = 59.25, df = 60, p-value = 0.5031
```

```
plot(bv.serial, names = "gold")
```

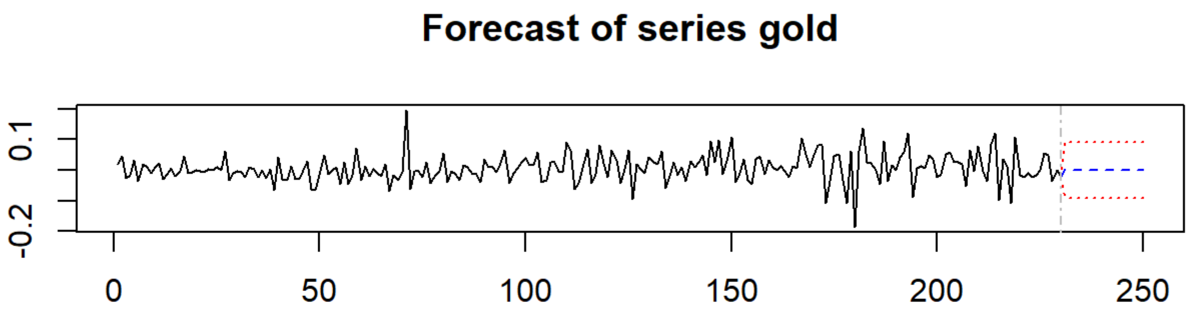


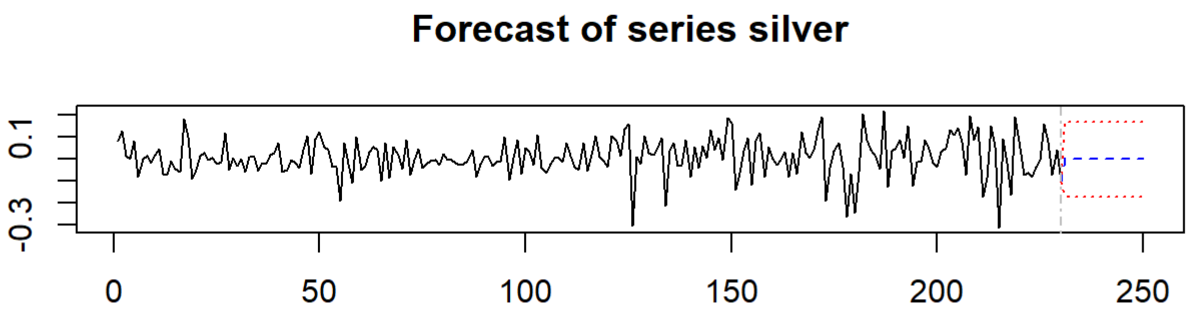

```
plot(bv.serial, names = "silver")
```



Forecasting usando el modelo VAR

```
fcast = predict(var.a, n.ahead = 20)
plot(fcast)
```





Forecast solo para gold

```
# guardar la serie de tiempo
gold = fcast$fcst[1]; gold
```

```
## $gold
##                fcst       lower      upper         CI
##  [1,]  2.209872e-03 -0.08879740 0.09321714 0.09100727
##  [2,] -2.827846e-04 -0.09203726 0.09147169 0.09175448
##  [3,]  3.446161e-05 -0.09173104 0.09179997 0.09176550
##  [4,] -4.228320e-06 -0.09176990 0.09176144 0.09176567
##  [5,]  5.183006e-07 -0.09176515 0.09176619 0.09176567
##  [6,] -6.354110e-08 -0.09176574 0.09176561 0.09176567
##  [7,]  7.789675e-09 -0.09176566 0.09176568 0.09176567
##  [8,] -9.549599e-10 -0.09176567 0.09176567 0.09176567
##  [9,]  1.170714e-10 -0.09176567 0.09176567 0.09176567
## [10,] -1.435213e-11 -0.09176567 0.09176567 0.09176567
## [11,]  1.759471e-12 -0.09176567 0.09176567 0.09176567
## [12,] -2.156988e-13 -0.09176567 0.09176567 0.09176567
## [13,]  2.644316e-14 -0.09176567 0.09176567 0.09176567
## [14,] -3.241746e-15 -0.09176567 0.09176567 0.09176567
## [15,]  3.974154e-16 -0.09176567 0.09176567 0.09176567
## [16,] -4.872034e-17 -0.09176567 0.09176567 0.09176567
## [17,]  5.972773e-18 -0.09176567 0.09176567 0.09176567
## [18,] -7.322201e-19 -0.09176567 0.09176567 0.09176567
## [19,]  8.976506e-20 -0.09176567 0.09176567 0.09176567
## [20,] -1.100457e-20 -0.09176567 0.09176567 0.09176567
```

```
# Extrayendo la columna de pronósticos
x = gold$gold[,1]; x
```

```
##  [1]  2.209872e-03 -2.827846e-04  3.446161e-05 -4.228320e-06  5.183006e-07
##  [6] -6.354110e-08  7.789675e-09 -9.549599e-10  1.170714e-10 -1.435213e-11
## [11]  1.759471e-12 -2.156988e-13  2.644316e-14 -3.241746e-15  3.974154e-16
## [16] -4.872034e-17  5.972773e-18 -7.322201e-19  8.976506e-20 -1.100457e-20
```

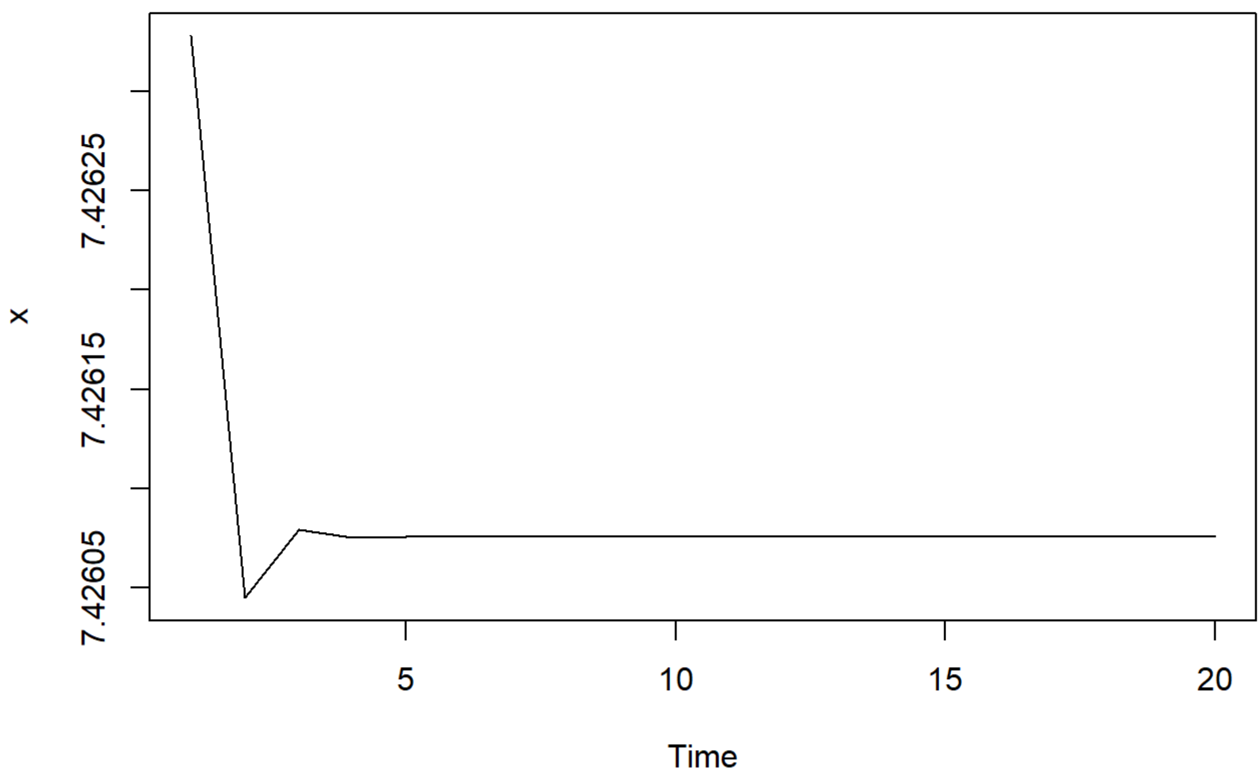
Invirtiendo la diferenciación

```
tail(data)
```

```
##            gold   silver
## [226,] 7.377496 3.311455
## [227,] 7.433862 3.469323
## [228,] 7.481798 3.545586
## [229,] 7.446702 3.473751
## [230,] 7.447728 3.516310
## [231,] 7.424118 3.411313
```

```
x = cumsum(x) + 7.424118

plot.ts(x)
```



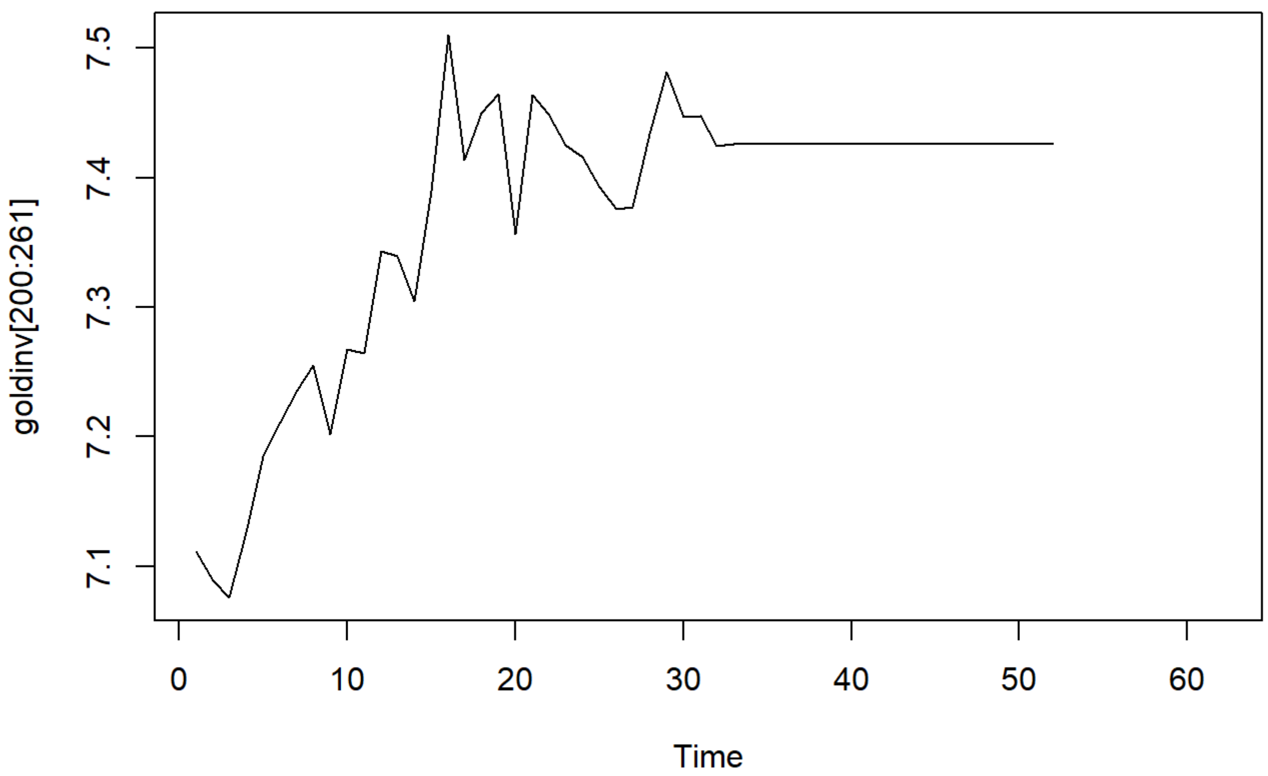
Combinando los datos reales y la predicción en una sola serie de tiempo

```
goldinv =ts(c(data[,1], x),
           start = c(1993,11), frequency = 12)
```

Dibujando todo

```
plot(goldinv)
```


```
plot.ts(goldinv[200:261])
```



Plot avanzado con separación visual entre lo real y lo pronosticado

```
library(lattice)
library(grid)
library(zoo)
```

Objeto zoo

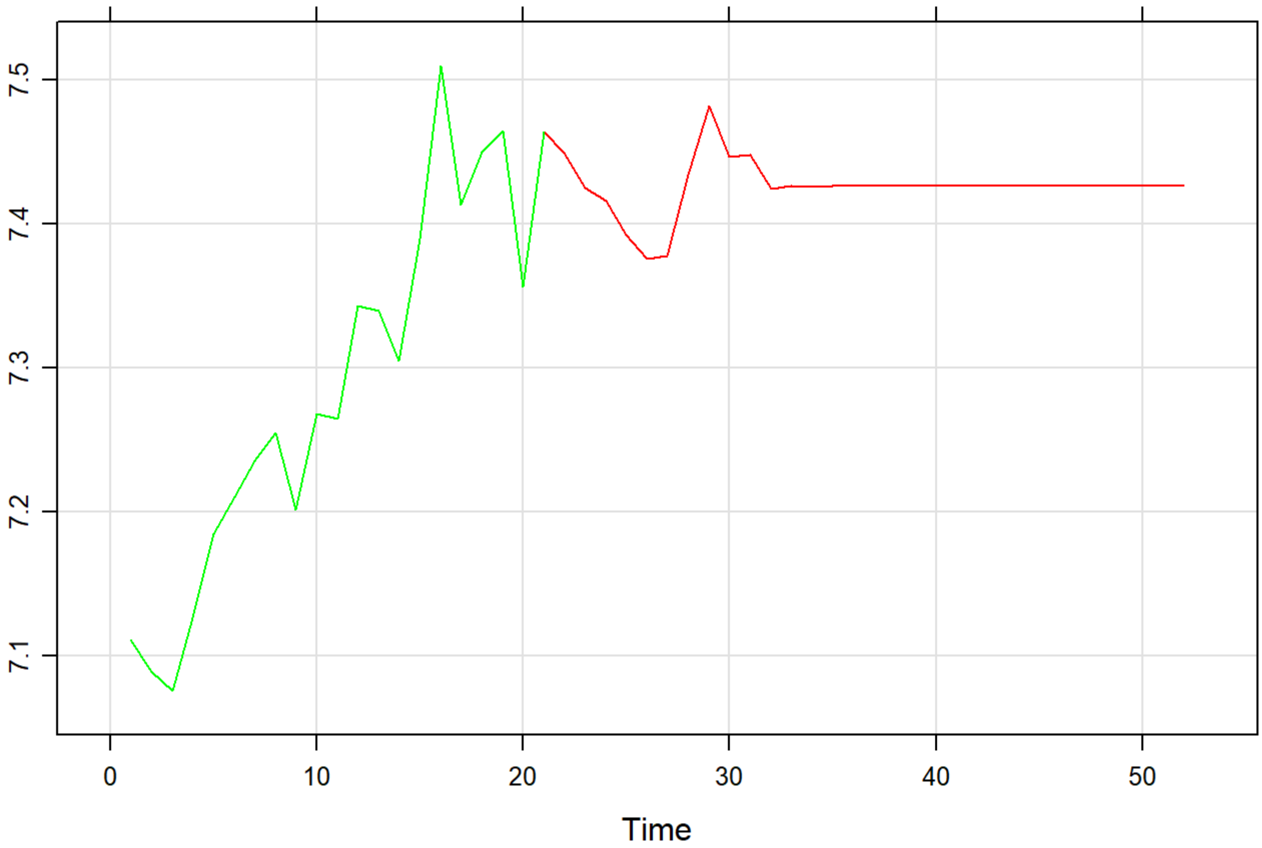
```
# realizado un zoom
xx = zoo(goldinv[200:251]) # reservamos 52 datos (30 reales + 22 pred)
```

```
xyplot(xx, grid=TRUE, panel = function(xx, y, ...){

  panel.xyplot(xx, y, col="red", ...)

  grid.clip(unit(21, "native"), just=c("right")) # añadimos 22-1 datos reales y el resto

  panel.xyplot(xx, y, col="green", ...) })
```



Forecast silver

```
fcast = predict(var.a, n.ahead = 30)
```

Solo para silver

```
silver = fcast$fcst[2]; silver
```

```
## $silver
##                fcst      lower     upper        CI
##  [1,]  2.835151e-03 -0.1655636 0.1712339 0.1683987
##  [2,] -5.215913e-04 -0.1703272 0.1692840 0.1698056
##  [3,]  6.092619e-05 -0.1697633 0.1698851 0.1698242
##  [4,] -7.521439e-06 -0.1698320 0.1698170 0.1698245
##  [5,]  9.211687e-07 -0.1698236 0.1698254 0.1698245
##  [6,] -1.129446e-07 -0.1698246 0.1698244 0.1698245
##  [7,]  1.384594e-08 -0.1698245 0.1698245 0.1698245
##  [8,] -1.697420e-09 -0.1698245 0.1698245 0.1698245
##  [9,]  2.080917e-10 -0.1698245 0.1698245 0.1698245
## [10,] -2.551059e-11 -0.1698245 0.1698245 0.1698245
## [11,]  3.127419e-12 -0.1698245 0.1698245 0.1698245
## [12,] -3.833997e-13 -0.1698245 0.1698245 0.1698245
## [13,]  4.700211e-14 -0.1698245 0.1698245 0.1698245
## [14,] -5.762130e-15 -0.1698245 0.1698245 0.1698245
## [15,]  7.063967e-16 -0.1698245 0.1698245 0.1698245
## [16,] -8.659929e-17 -0.1698245 0.1698245 0.1698245
## [17,]  1.061647e-17 -0.1698245 0.1698245 0.1698245
## [18,] -1.301504e-18 -0.1698245 0.1698245 0.1698245
## [19,]  1.595553e-19 -0.1698245 0.1698245 0.1698245
## [20,] -1.956037e-20 -0.1698245 0.1698245 0.1698245
## [21,]  2.397964e-21 -0.1698245 0.1698245 0.1698245
## [22,] -2.939736e-22 -0.1698245 0.1698245 0.1698245
## [23,]  3.603910e-23 -0.1698245 0.1698245 0.1698245
## [24,] -4.418141e-24 -0.1698245 0.1698245 0.1698245
## [25,]  5.416332e-25 -0.1698245 0.1698245 0.1698245
## [26,] -6.640043e-26 -0.1698245 0.1698245 0.1698245
## [27,]  8.140228e-27 -0.1698245 0.1698245 0.1698245
## [28,] -9.979349e-28 -0.1698245 0.1698245 0.1698245
## [29,]  1.223398e-28 -0.1698245 0.1698245 0.1698245
## [30,] -1.499801e-29 -0.1698245 0.1698245 0.1698245
```

Extrayendo la columna de pronósticos

```
y = silver$silver[,1]; y
```

```
##  [1]  2.835151e-03 -5.215913e-04  6.092619e-05 -7.521439e-06  9.211687e-07
##  [6] -1.129446e-07  1.384594e-08 -1.697420e-09  2.080917e-10 -2.551059e-11
## [11]  3.127419e-12 -3.833997e-13  4.700211e-14 -5.762130e-15  7.063967e-16
## [16] -8.659929e-17  1.061647e-17 -1.301504e-18  1.595553e-19 -1.956037e-20
## [21]  2.397964e-21 -2.939736e-22  3.603910e-23 -4.418141e-24  5.416332e-25
## [26] -6.640043e-26  8.140228e-27 -9.979349e-28  1.223398e-28 -1.499801e-29
```

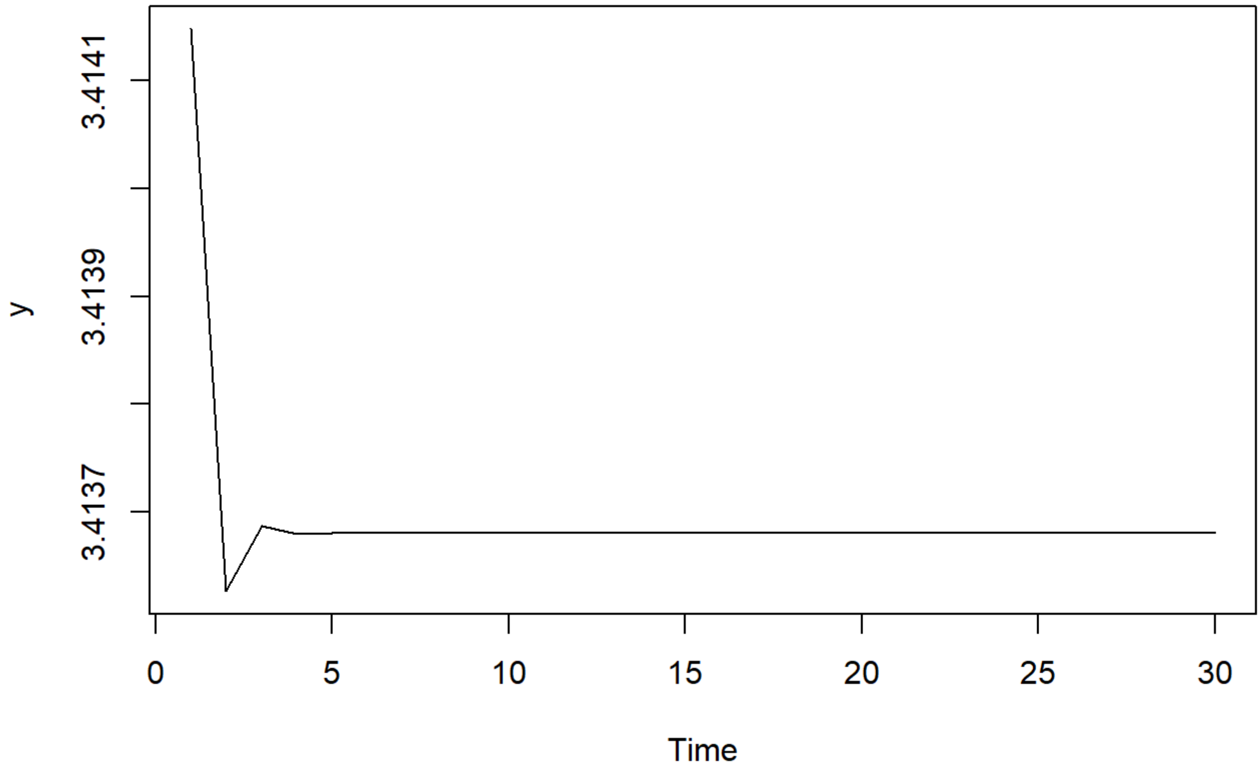
Invirtiendo la diferenciación

```
tail(data)
```

```
##             gold   silver
## [226,] 7.377496 3.311455
## [227,] 7.433862 3.469323
## [228,] 7.481798 3.545586
## [229,] 7.446702 3.473751
## [230,] 7.447728 3.516310
## [231,] 7.424118 3.411313
```

```
y = cumsum(y) + 3.411313

plot.ts(y)
```

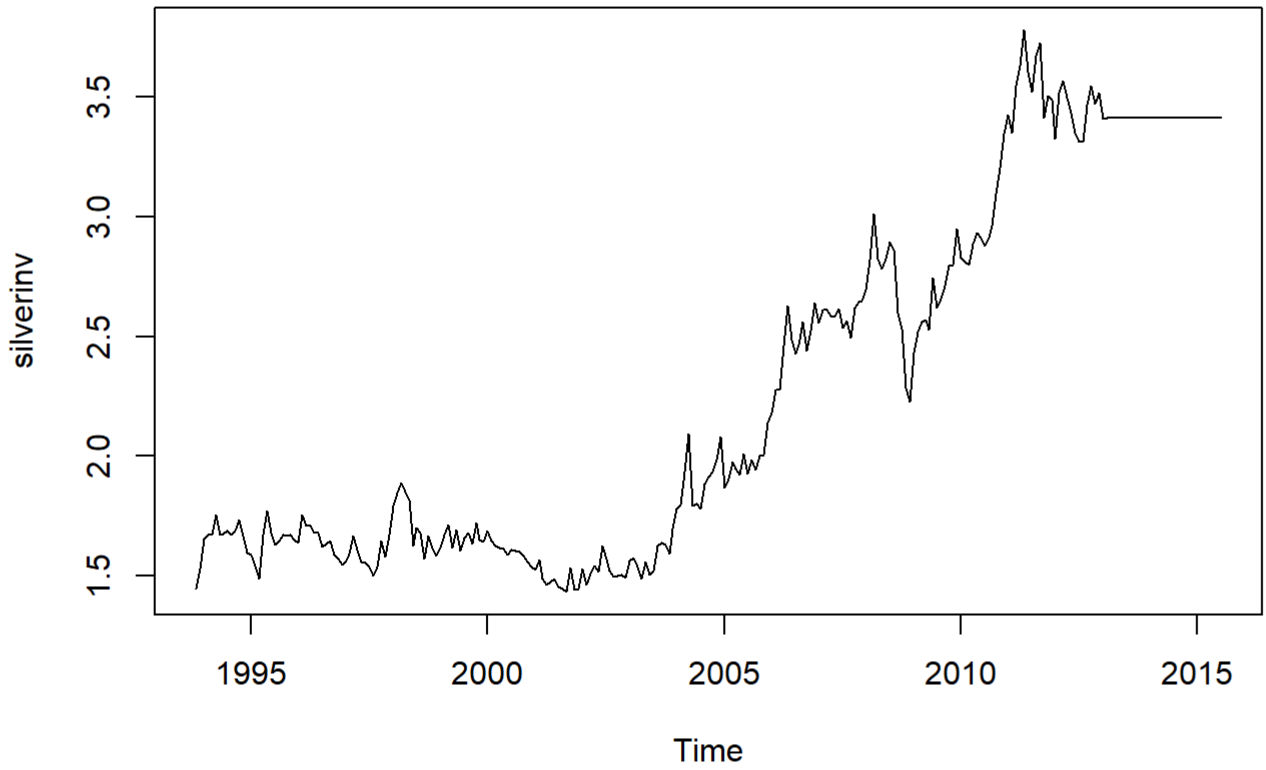


Combinando los datos reales y la predicción en una sola serie de tiempo

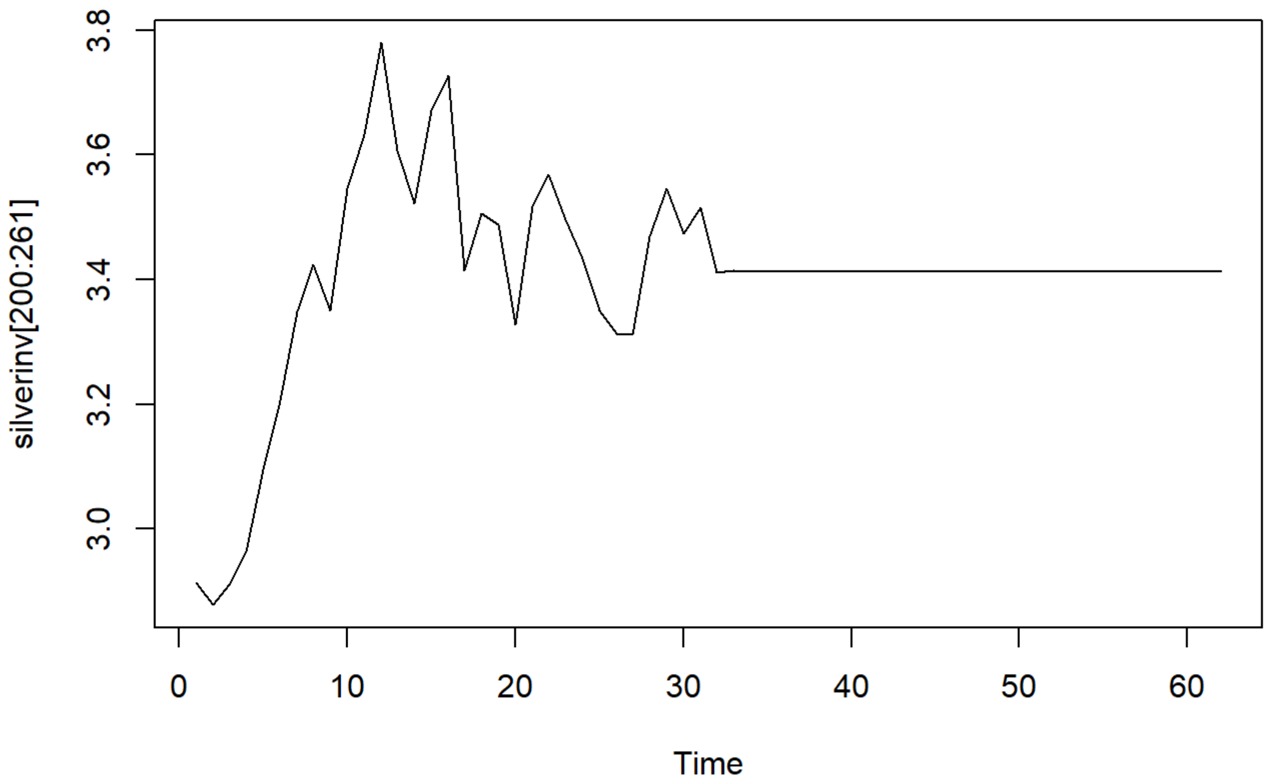
```
silverinv =ts(c(data[,2], y),
            start = c(1993,11), frequency = 12)
```

Dibujando todo

```
plot(silverinv)
```




```
plot.ts(silverinv[200:261])
```



Plot avanzado con separación visual entre lo real y lo pronosticado

```
library(lattice)
library(grid)
library(zoo)
```

```
## Objeto zoo
xx = zoo(silverinv[200:261])
```

```
# En el parámetro grid.clip ponemos la cantidad de observaciones que son reales dentro de las
# que hemos elegido. Hemos cogido 62 de las que 30 son pronósticos, así que grid.clip sería 3
2-1

xyplot(xx, grid=TRUE, panel = function(xx, y, ...){

  panel.xyplot(xx, y, col="red", ...)

  grid.clip(unit(31, "native"), just=c("right"))

  panel.xyplot(xx, y, col="green", ...) })
```