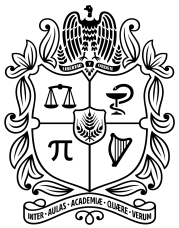
Series de tiempo univariadas - Presentación 19

Mauricio Alejandro Mazo Lopera

Universidad Nacional de Colombia
Facultad de Ciencias
Escuela de Estadística
Medellín



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Backtesting o validación de modelos en series de tiempo

Hasta ahora hemos visto algunos criterios de selección de modelos que permiten seleccionar el "mejor" tales como: AIC, BIC, AICC, entre otros.

Backtesting o validación de modelos en series de tiempo

Hasta ahora hemos visto algunos criterios de selección de modelos que permiten seleccionar el "mejor" tales como: AIC, BIC, AICC, entre otros. Estos criterios se suelen utilizar con la BD completa y no evalúan cómo se comporta el modelo con respecto a los pronósticos.

El **backtesting** es un método de valiación en series de tiempo utilizado para comparar los pronósticos que realiza un modelo versus los valores reales.

Para ver cómo funciona este método definimos primero algunas medidas de precisión:

Backtesting o validación de modelos en series de tiempo

- **Media del Error Absoluto** (MAE):

$$MAE = \sum_{t=1}^{m} \frac{|X_t - \widehat{X}_t|}{m}$$

- **Media Cuadrática del Error** (MSE):

$$MSE = \sum_{t=1}^{m} \frac{(X_t - \widehat{X}_t)^2}{m}$$

- **Media del error porcentual absoluto** (MAPE):

$$MAPE = 100 * \frac{1}{m} \sum_{t=1}^{m} \frac{|X_t - \widehat{X}_t|}{|X_t|}$$

donde $\widehat{X}_t$ es la predicción para el t-ésimo individuo.

Backtesting o validación de modelos en series de tiempo

Estas medidas podemos programarlas en R como:

```
MAE <- function(real, pred){
  mean( abs (real-pred) )
}

MSE <- function(real, pred){
  mean( (real-pred)^2 )
}

MAPE <- function(real, pred){
 100 * mean( abs( (real-pred)/real ) )
}
```

Backtesting o validación de modelos en series de tiempo

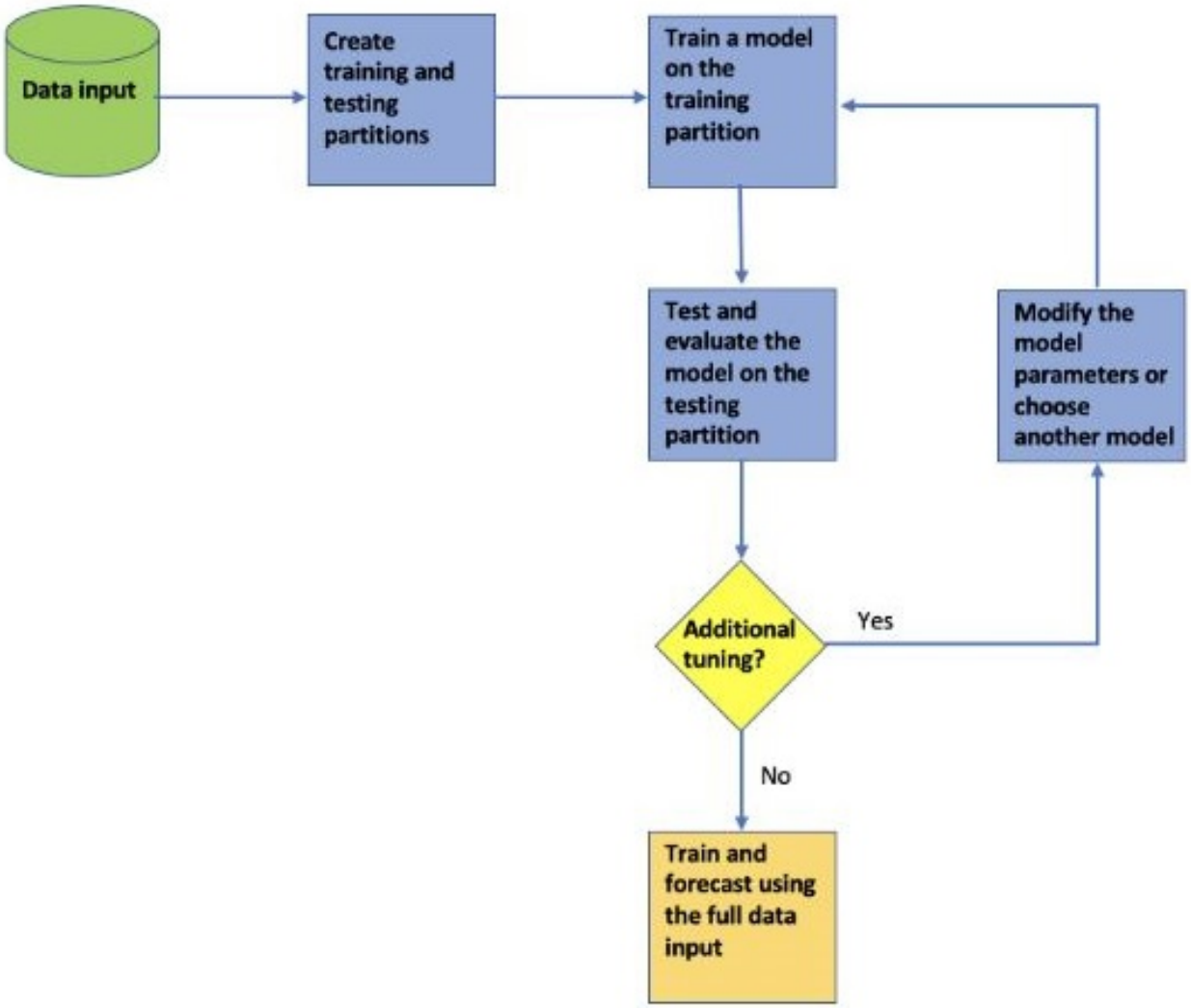
En el paquete **forecast** existe una función conocida como **accuracy** (escriba ?accuracy para pedir ayuda en R) que contiene estas y otras medidas tales como:

- ME: Mean Error
- RMSE: Root Mean Squared Error
- MAE: Mean Absolute Error
- MPE: Mean Percentage Error
- MAPE: Mean Absolute Percentage Error
- MASE: Mean Absolute Scaled Error

NOTA: Con cualquiera de estas medidas, lo ideal es que sean lo más cercanas a cero que sea posible.

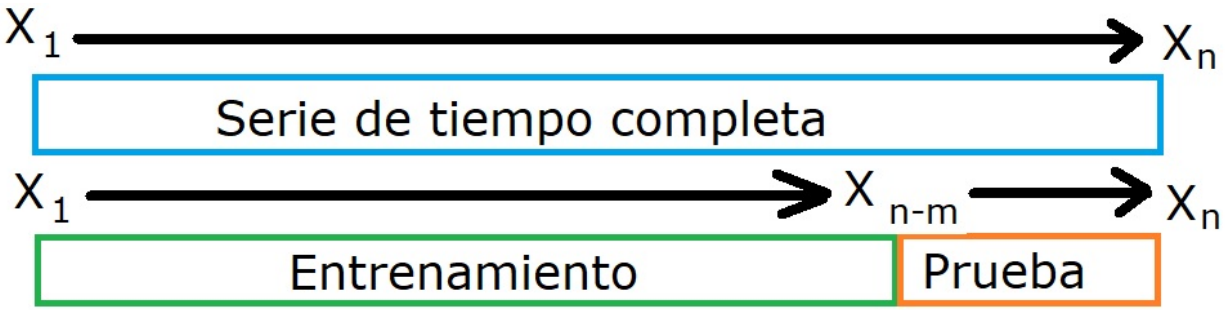
Backtesting: Diagrama de Flujo

Tomado de **Rami Krispin - Hands-On Time Series Analysis With R_ Perform Time Series Analysis And Forecasting Using R-Packt Publishing (2019)**



Backtesting: Una única partición

El método más simple de validación consiste en particionar la serie en solo dos conjuntos de datos: Entrenamiento (Train) y Prueba (Test):



Veamos un ejemplo con una BD ya antes estudiada:

```
require(TSstudio)
data(USgas)
ts_info(USgas)
```

```
##  The USgas series is a ts object with 1 variable and 238 observation
##  Frequency: 12
##  Start time: 2000 1
##  End time: 2019 10
```

Backtesting: Una única partición

Como vemos, la serie tiene datos mensuales de casi 20 años.

Backtesting: Una única partición

Como vemos, la serie tiene datos mensuales de casi 20 años. Particionamos la BD, dejando un año por fuera como datos de prueba para realizar pronósticos y evaluar la precisión. Usamos la función **window** del paquete **stats**:

```
train <- window(USgas, start=time(USgas)[1],
                end = time(USgas)[length(USgas) - 12])

test <- window(USgas, start = time(USgas)[length(USgas)
                                          - 12 + 1],
               end = time(USgas)[length(USgas)])
```

Vemos la información de los datos de entrenamiento y prueba:

Backtesting: Una única partición

```
ts_info(train)
```

```
##  The train series is a ts object with 1 variable and 226 observation
##  Frequency: 12
##  Start time: 2000 1
##  End time: 2018 10
```

```
ts_info(test)
```

```
##  The test series is a ts object with 1 variable and 12 observations
##  Frequency: 12
##  Start time: 2018 11
##  End time: 2019 10
```

Backtesting: Una única partición

La partición anterior también es posible con la función **ts_split** del paquete **TSstudio**:

```
USgas_partitions <- ts_split(USgas, sample.out = 12)
train <- USgas_partitions$train
test <- USgas_partitions$test
```

```
ts_info(train)
```

```
##  The train series is a ts object with 1 variable and 226 observation
##  Frequency: 12
##  Start time: 2000 1
##  End time: 2018 10
```

```
ts_info(test)
```

```
##  The test series is a ts object with 1 variable and 12 observations
##  Frequency: 12
##  Start time: 2018 11
##  End time: 2019 10
```

Backtesting: Una única partición

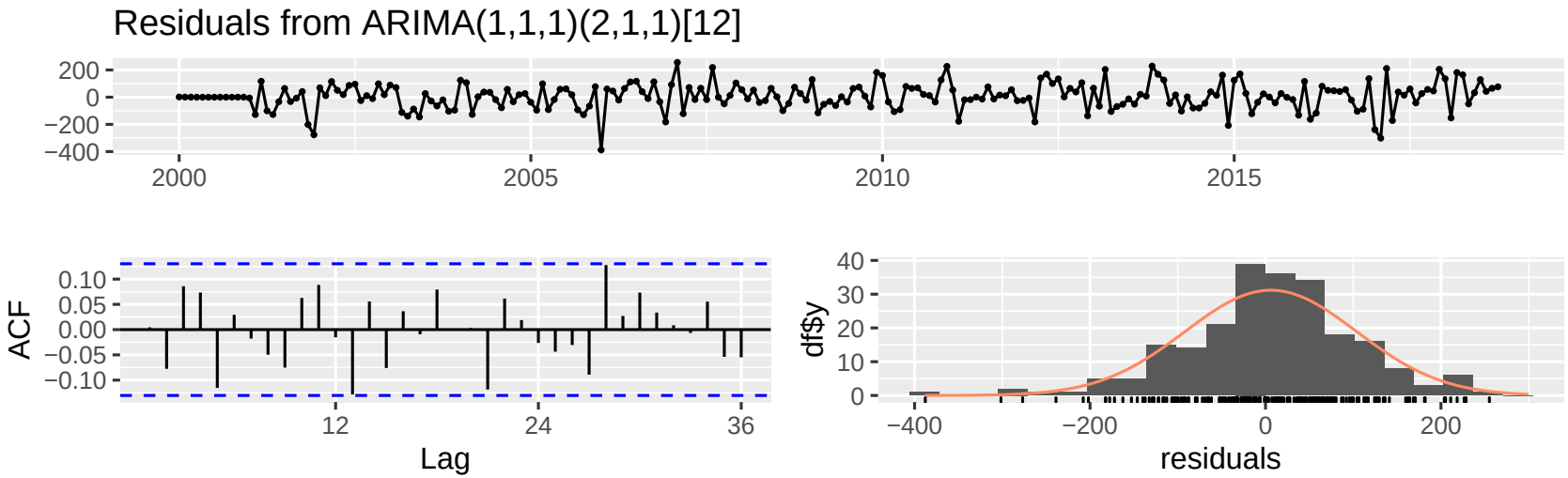
Usamos el **auto.arima**:

```
require(forecast)
modelo1.train <- auto.arima(train, stepwise = FALSE,
                            approximation = FALSE)
modelo1.train
```

```
## Series: train
## ARIMA(1,1,1)(2,1,1)[12]
##
## Coefficients:
##          ar1      ma1    sar1     sar2     sma1
##       0.4247  -0.9180  0.0132  -0.2639  -0.7449
## s.e.  0.0770   0.0376  0.0894   0.0834   0.0753
##
## sigma^2 = 10405:  log likelihood = -1292.96
## AIC=2597.91   AICc=2598.32   BIC=2618.08
```

Backtesting: Una única partición

```
require(forecast)
checkresiduals(modelo1.train)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,1)(2,1,1)[12]
## Q* = 25.336, df = 19, p-value = 0.1498
##
## Model df: 5.   Total lags used: 24
```

Backtesting: Una única partición

```
require(tseries)
shapiro.test(modelo1.train$residuals)
```

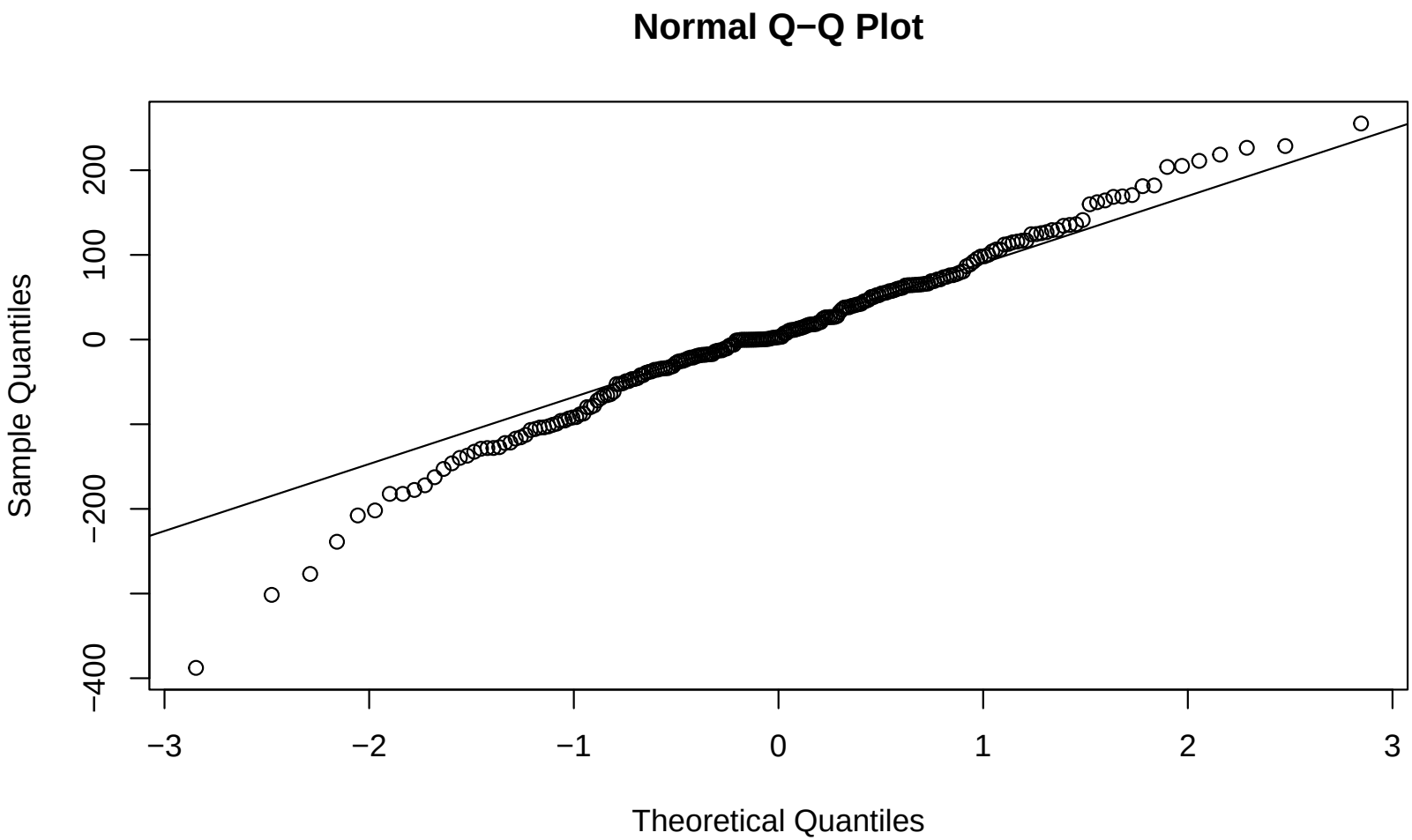
```
##
##  Shapiro-Wilk normality test
##
## data:  modelo1.train$residuals
## W = 0.98267, p-value = 0.007231
```

```
jarque.bera.test(modelo1.train$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  modelo1.train$residuals
## X-squared = 18.622, df = 2, p-value = 9.041e-05
```

Backtesting: Una única partición

```
qqnorm(modelo1.train$residuals)
qqline(modelo1.train$residuals)
```



Backtesting: Una única partición

Asumiendo que los residuales del modelo provienen de una distribución normal, entonces evaluamos la precisión del modelo mediante la función **accuracy** del paquete **forecast**:

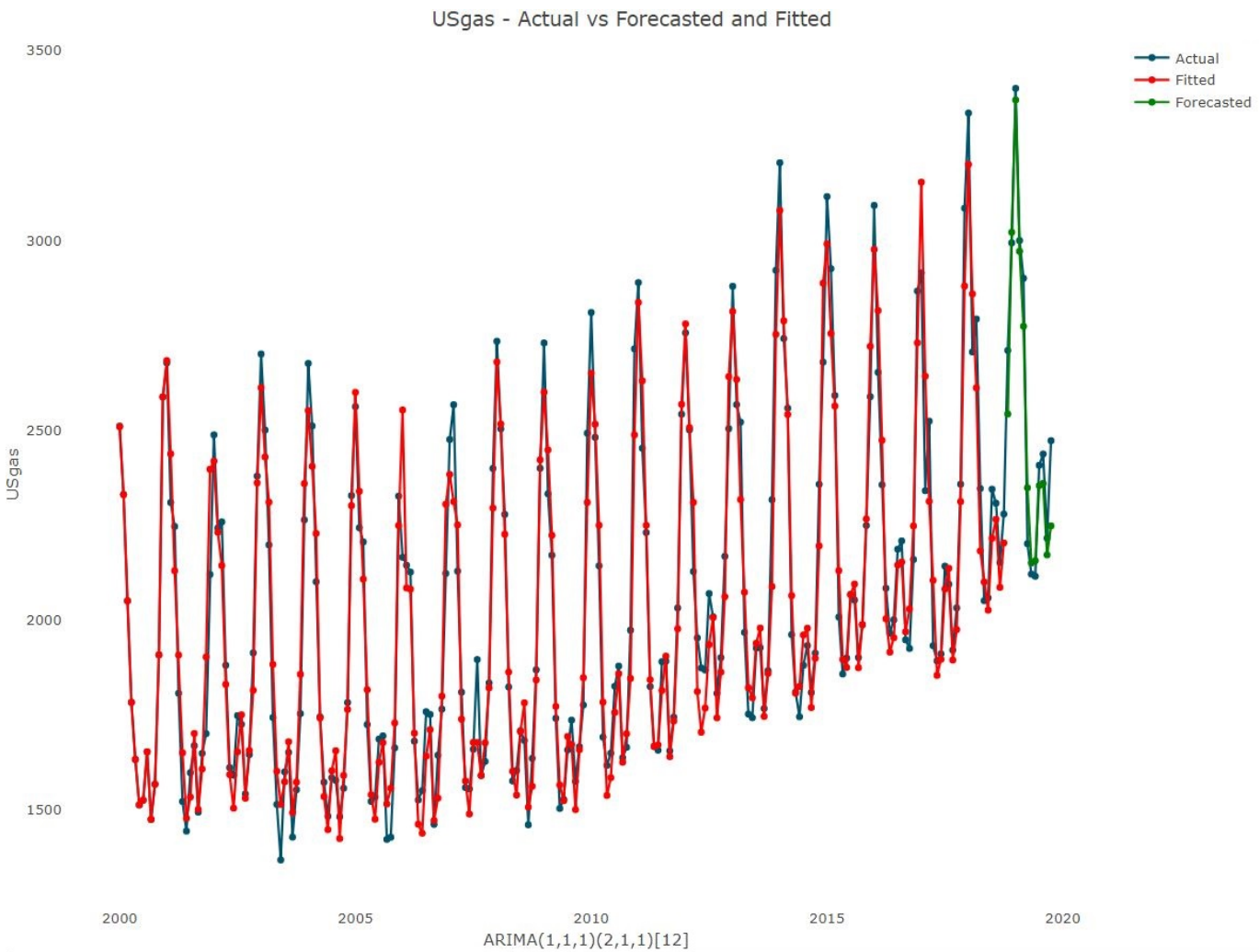
```
fore1 <- forecast(modelo1.train, h=12)
accuracy(fore1, test)
```

```
##                       ME      RMSE      MAE        MPE     MAPE      MASE
## Training set  6.081099  97.85701 73.36854 0.1298714 3.517097 0.6371821
## Test set     42.211253 104.79281 83.09943 1.4913412 3.314280 0.7216918
##                     ACF1 Theil's U
## Training set  0.004565602        NA
## Test set     -0.049999868 0.3469228
```

Cada uno de estos valores sirve para ser comparado con los obtenidos por algún otro modelo. A este proceso se le conoce como **benchmarking** entre modelos.

Backtesting: Una única partición

```
test_forecast(actual = USgas, forecast.obj = fore1,
              test = test)
```

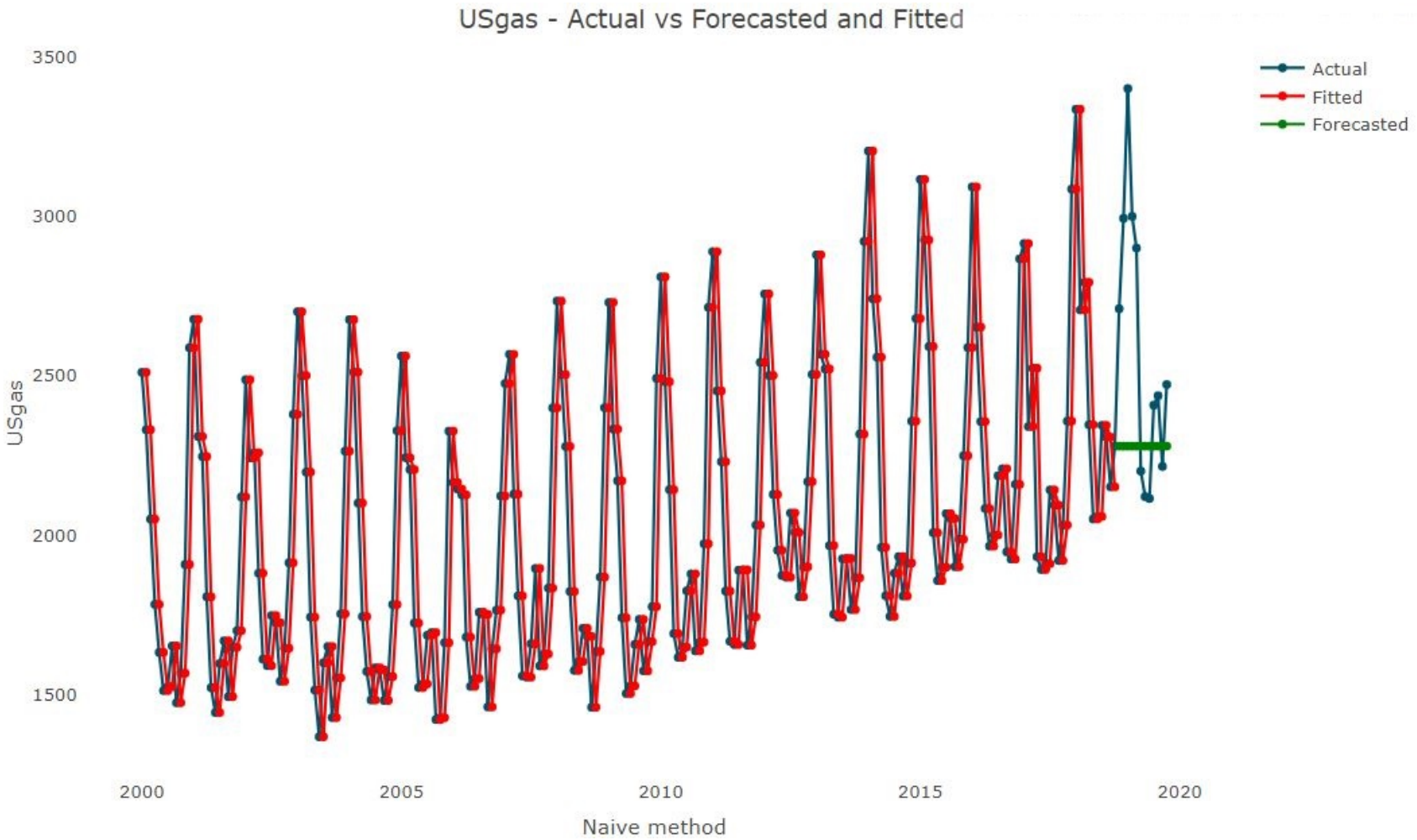


Benchmarking de modelos

Para ver cómo funciona el Benchmarking, consideremos el modelo que ajustamos antes y comparemos éste con un método conocido como "ingenuo" (**naive** en inglés) que asume que el último valor es el "mejor" pronóstico para todos los valores futuros. Dichas predicciones están en la función **naive** del paquete **forecast**:

```
naive_model1 <- naive(train, h = 12)
test_forecast(actual = USgas,
              forecast.obj = naive_model1,
              test = test)
```

Benchmarking de modelos



Benchmarking de modelos

Evaluamos la precisión de este modelo:

```
accuracy(naive_model1, test)
```

```
##                       ME     RMSE      MAE         MPE     MAPE     MASE
## Training set  -1.028444 285.6607 228.5084 -0.9218463 10.97123 1.984522
## Test set     301.891667 499.6914 379.1417  9.6798015 13.28187 3.292723
##                    ACF1 Theil's U
## Training set 0.3761105        NA
## Test set     0.7002486  1.499679
```

Comparamos con los del primer modelo, el cual es mucho mejor:

```
accuracy(fore1, test)
```

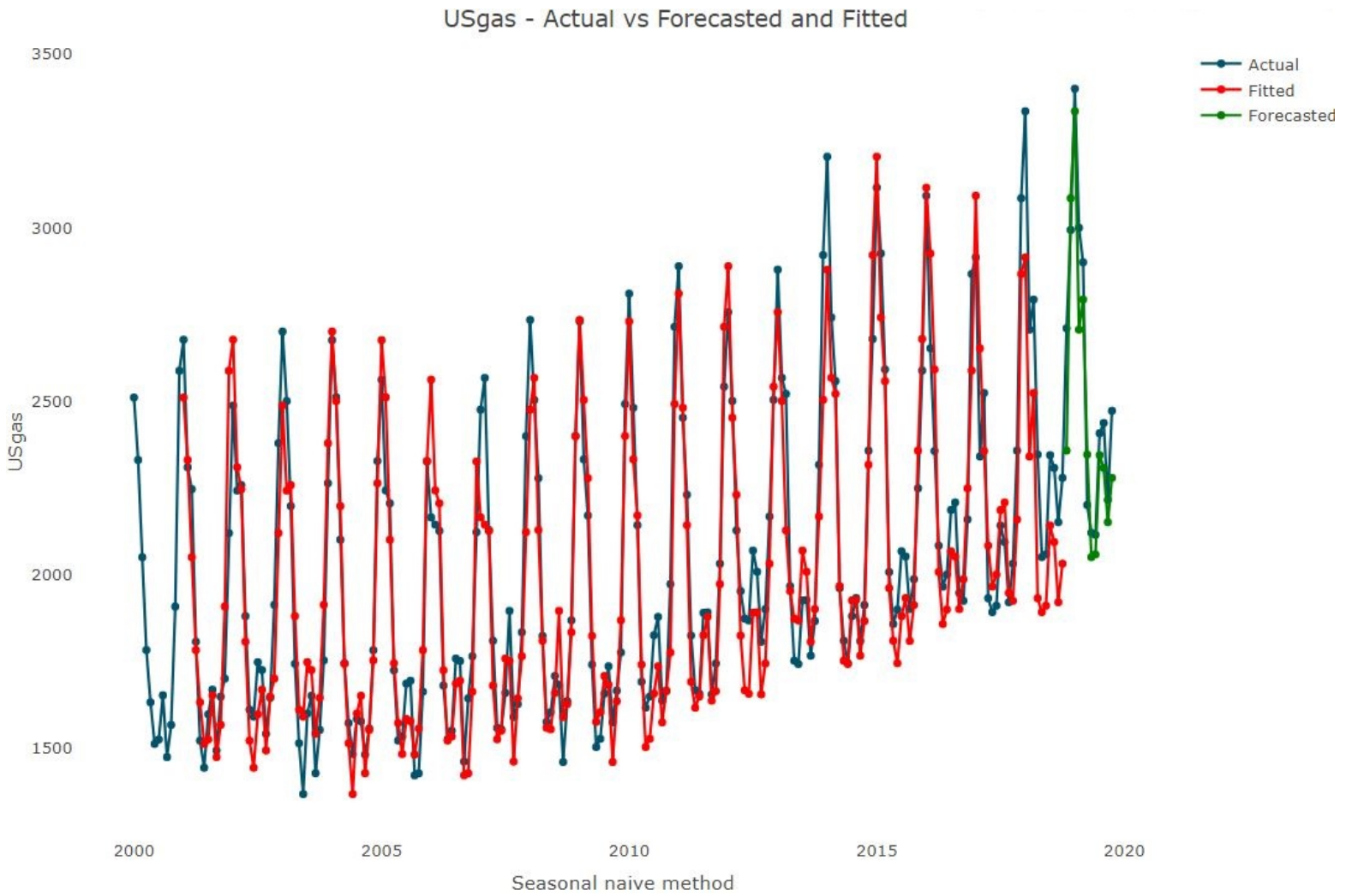
```
##                       ME      RMSE      MAE        MPE     MAPE      MASE
## Training set  6.081099  97.85701 73.36854 0.1298714 3.517097 0.6371821
## Test set     42.211253 104.79281 83.09943 1.4913412 3.314280 0.7216918
##                     ACF1 Theil's U
## Training set  0.004565602        NA
## Test set     -0.049999868 0.3469228
```

Benchmarking de modelos

En el caso de estacionalidad se usa la función **snaive** que tiene en cuenta el componente estacional para realizar los pronósticos. Dichas predicciones están en la función **snaive** del paquete **forecast**:

```
snaive_model1 <- snaive(train, h = 12)
test_forecast(actual = USgas,
              forecast.obj = snaive_model1,
              test = test)
```

Benchmarking de modelos



Benchmarking de modelos

Evaluamos la precisión de este modelo:

```
accuracy(snaive_model1, test)
```

```
##                     ME     RMSE      MAE      MPE     MAPE     MASE        ACF
## Training set 33.99953 148.7049 115.1453 1.379869 5.494048 1.000000  0.485950
## Test set     96.45000 164.6967 135.8833 3.612060 5.220458 1.180103 -0.212092
##              Theil's U
## Training set        NA
## Test set     0.4289964
```

Comparamos con los del primer modelo, el cual también es mucho mejor:

```
accuracy(fore1, test)
```

```
##                     ME      RMSE      MAE       MPE     MAPE      MASE
## Training set  6.081099  97.85701 73.36854 0.1298714 3.517097 0.6371821
## Test set     42.211253 104.79281 83.09943 1.4913412 3.314280 0.7216918
##                     ACF1 Theil's U
## Training set  0.004565602        NA
## Test set     -0.049999868 0.3469228
```

Actividad de clase:

Considere las bases de datos del petróleo brent (con nombre **petroleo_brent_historico.csv**) y de la tasa representativa del mercado (**trm_historico.csv**) que se encuentran en la carpeta DATOS del Google Drive.

1. Lea ambas bases de datos.
2. Una las dos bases de datos en una sola BD.
3. Realice un gráfico donde aparezcan ambas series de tiempo.
4. Divida la BD en dos conjuntos: entrenamiento y prueba.
5. Encuentre un modelo con la función **auto.arima**.
6. Realice un diagnóstico del modelo y saque conclusiones.
7. Evalúe el modelo ajustado con los datos de prueba y compare con un modelo "naive".