Series Temporales - ARIMA

Motivación

Una serie temporal es una secuencia de datos discretos en el tiempo, equidistantes. El forecasting de series temporales se refiere al uso de un modelo para predecir valores futuros basados en los valores previamente observados. En anteriores trabajos se detalló el uso de modelos lineales generalizados aditivos para poder trabajar con series temporales, como por ejemplo *Prophet*, por lo que en el siguiente, se presentará la teoría necesaria para poder enteder e implementar los modelos AR, MA y ARIMA, en conjunto con un caso práctico con código.

El caso que se analizará es un dataset publicado en Rpubs de la ocupación de pasajes aéroes (cantidad de pasajeros que volaron) por mes desde 1949 a 1960. Se tomará como referencias y bases de investigación a:

- "Air Passengers Occupancy Prediction Using Arima Model, International Journal of Applied Engineering Research"
- "Forecasting: Principles and Practice", Hyndam and Athanasopoulos
- "Kaggle - Time Series Analysis , Dileep 2019"

```
#install.packages("tsibble")
library(tsibble)
#install.packages("feasts")
library(feasts)
library(tidyverse)
library(openintro)
library(GGally)
library(corrr)
library(knitr)
library(tidymodels)
library(gridExtra)
library(rsample)
library(ggplot2)
library(GGally)
library(RColorBrewer)
library(ggcorrplot)
library(forcats)
library(robustbase)
library(fpp3)
library(tseries)
library(slider)
library(forecast)
```

Primeros Pasos

Levantamos el dataset a analizar.

El dataset cuenta con dos variables, la temporal referida al año y mes y luego la variable de cantidad de pasajeros que representa nuestro y.

```
df <-  read.csv("~/Documents/Data-Mining/EEA/TP2/data/AirPassengers.csv")

head(df)
```

```
##     Month X.Passengers
## 1 1949-01          112
## 2 1949-02          118
## 3 1949-03          132
## 4 1949-04          129
## 5 1949-05          121
## 6 1949-06          135
```

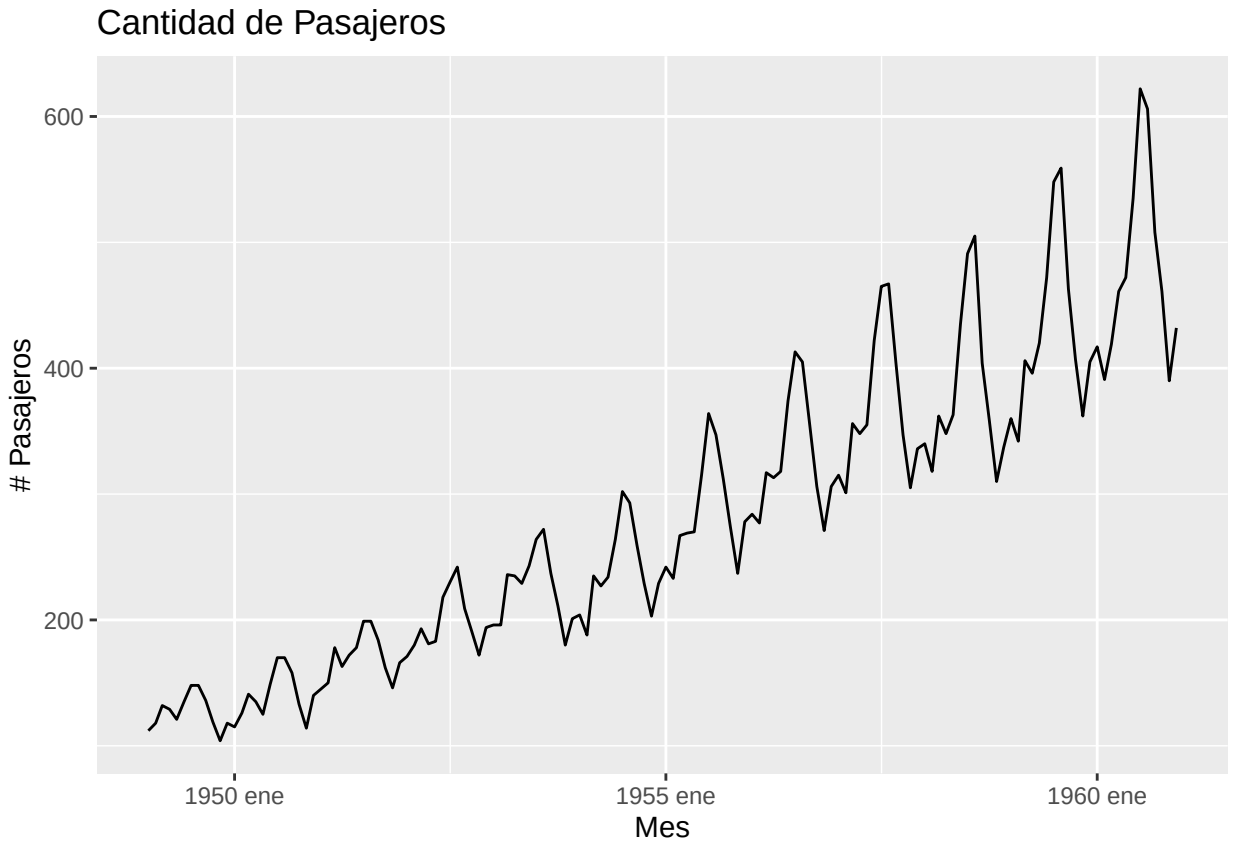
Como primer paso, es necesario transformar el dataset en un objeto tsibble para poder analizarlo como serie temporal.

```
df <- df  %>% mutate(Month = yearmonth(Month)) %>% as_tsibble(index= Month)
```

Visualización

Antes de comenzar algún análisis visualizemos el grafico de cantidad de pasajeros en función del tiempo. Del gráfico se pueden notar varios puntos: *Existe una tendencia creciente, es decir que vemos que el valor medio de cantidad de pasajeros aumenta a medida que aumenta el tiempo.* La variación aumenta a medida que crece el tiempo.

```
autoplot(df, X.Passengers) +
  labs(title = "Cantidad de Pasajeros", y = "# Pasajeros", x = "Mes")
```

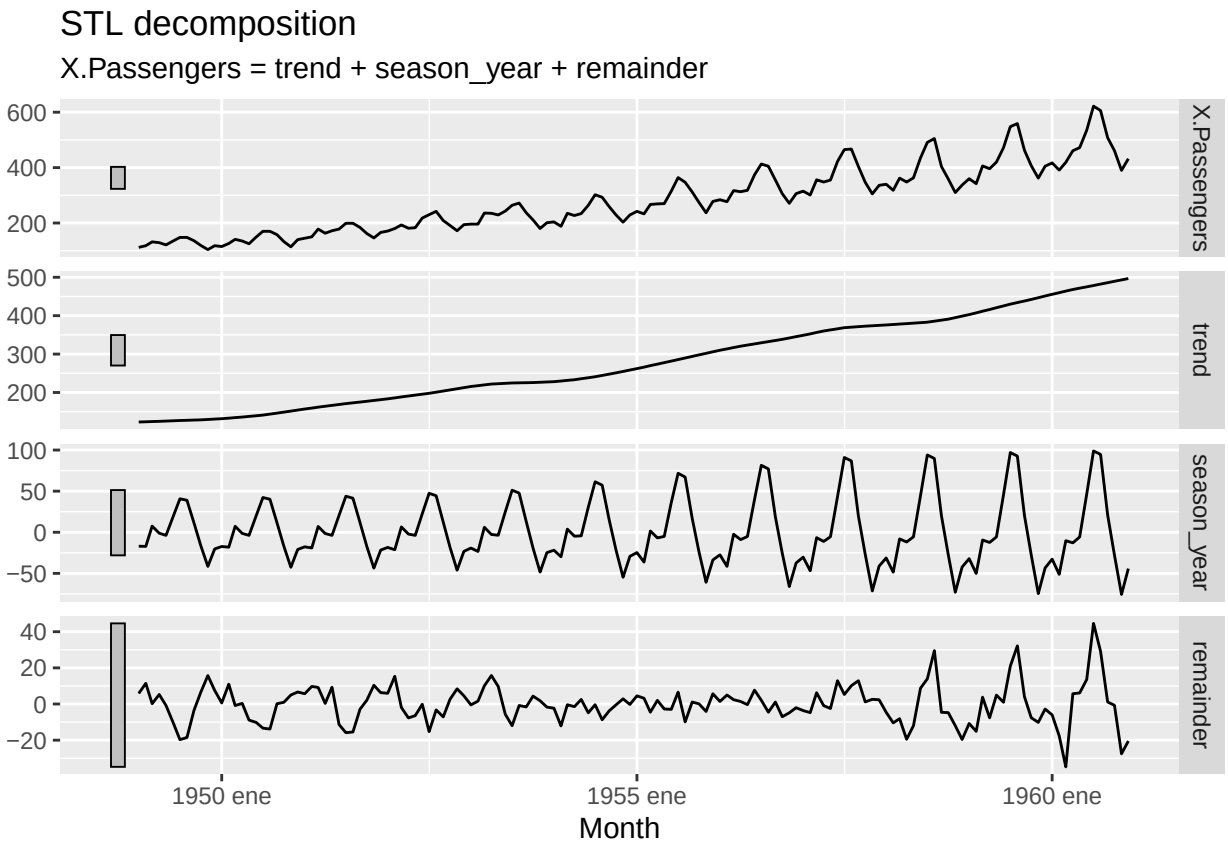


Componentes de una Serie Temporal Si bien en este trabajo no se utilizará GAM y Prophet como modelo, es util para poder enteder las componentes. Estos representan la serie como un modelo aditivo propuesto por

Harvey & Peters (1990): \begin{equatio} $y(t) = S(t) + T(t) + R(t) + \epsilon_t$: \end{equation} donde, $y(t)$ *es la variable a predecir,* $S(t)$ es la componente de seasonality (estacionalidad). Es un patrón que ocurre por factores estacionales, siempre de un período fijo y conocido. No confundir con **ciclo** que ocurre cuando existe un patrón de datos de aumento y disminución que no son de una frecuencia fija. $T(t)$ *es la componente de tendencia. Representa un aumento o disminución a largo plazo en los datos.* $R(t)$ es la componente remanente, que puede representar efectos atípicos. $*\epsilon_t$ es el error del modelo en la predicción de y_t.

Estas componentes podemos verlas utilizando los siguientes comandos:´

```
dcmp <- df %>%
  model(stl = STL(X.Passengers))

#components(dcmp) %>%
#  as_tsibble() %>% autoplot()
#  autoplot(X.Passengers, colour="gray") +
#  geom_line(aes(y=trend), colour = "#D55E00") +
#  labs(
#    y = "Cantidad de Pasajeros", x= "Mes")

components(dcmp) %>% autoplot()
```



Como se puede visualizar en el anterio gráfico, los puntos mencionados anteriormente se comprueban. La tendencia es creciente, y vemos en la seasonality que la varianza aumenta a lo largo del tiempo. Esto luego nos interesará más adelante cuando hablemos de **estacionariedad** (*stacionarity*).

Muchas veces, para poder eliminar parte de la aleatoriedad de los datos se los suaviza con distintas técnicas. En anteriores trabajos se vio **loess**, que suaviza por medio de un ajuste por regresión polinómica local. En este, vamos a proponer otro método de suavizado: **Medias Móviles** (*moving average smoothing*). La idea detrás sería utilizar:

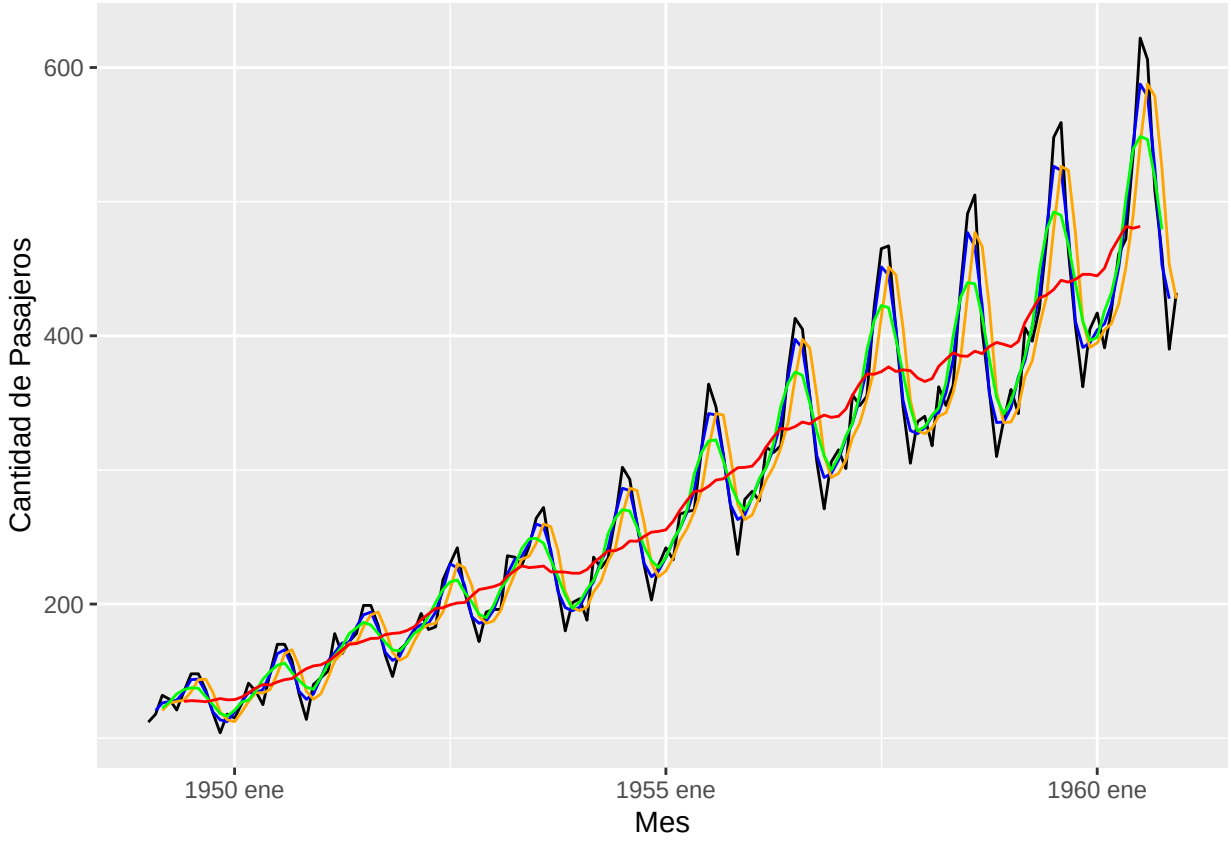
$$\hat{T} = \frac{1}{2k+1} \sum_{j=-k}^{k} y_t + j \tag{1}$$

De forma análoga existe el **Rolling Average**, que se diferencia en un suavizado tomando ventanas no simétricas, por ejemplo solamente tomando *j=0,-1,-2*. Visualizemos estos suavizados.

```
df_movil <- df %>%
    mutate(
    "k_1" = slider::slide_dbl(X.Passengers, mean,
                .before = 1, .after = 1, .complete = TRUE)
  )

df_movil <- df_movil %>%
    mutate(
    "k_3_rolling" = slider::slide_dbl(X.Passengers, mean,
                .before = 2, .after = 0, .complete = TRUE)
  )

df_movil <- df_movil %>%
    mutate(
   "k_2" = slider::slide_dbl(X.Passengers, mean,
                .before = 2, .after = 2, .complete = TRUE)
  )

df_movil <- df_movil %>%
    mutate(
   "k_5" = slider::slide_dbl(X.Passengers, mean,
                .before = 5, .after = 5, .complete = TRUE)
  )
```

```
  colors <- c("X.Passengers" = "black", "k_1" = "blue", "k_3_rolling" = "orange", "k_2"="green","k_5"="

df_movil %>%
  autoplot(X.Passengers, colour = "black") +
  geom_line(aes(y = k_1), colour = "blue") +
  geom_line(aes(y = k_3_rolling), colour = "orange") +
  geom_line(aes(y = k_2), colour = "green") +
  geom_line(aes(y = k_5), colour = "red") +
  labs(y  = "Cantidad de Pasajeros", x= "Mes")+
  scale_color_manual(values=colors)#,labels=c("Pasajeros","k_1","k_3_rolling","k_2","k_5"))
```

```
## Warning: Removed 2 row(s) containing missing values (geom_path).

## Warning: Removed 2 row(s) containing missing values (geom_path).

## Warning: Removed 4 row(s) containing missing values (geom_path).

## Warning: Removed 10 row(s) containing missing values (geom_path).
```



Conceptos Modelos AR, MA, ARIMA

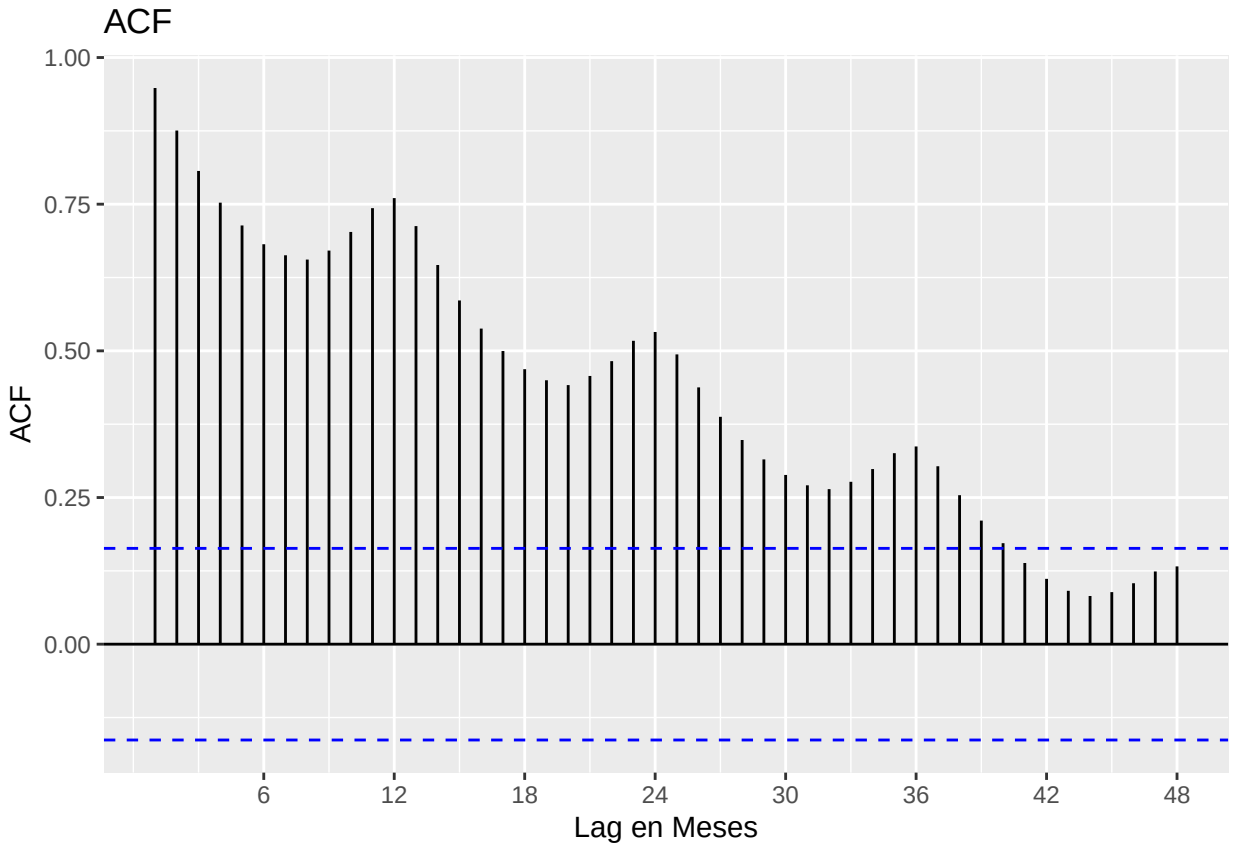
Autocorrelacion

La autocorrelacion cuantifica la relación lineal entre los valores de una serie y sus k valores anteriores. Cuantifica la similaridad entre la serie y una versión de ella misma en una ventana temporal anterior sobre sucesivos intervalos. Matemáticamente se expresa como:

$$r_k = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2} T : lengthdelaserie, \bar{y} : mediadeyk : lag \tag{2}$$

La función de autocorrelación (**ACF**) generaliza este coeficiente. Los efectos de tendencia y estacionalidad influyen fuertemente en la autocorrelación, y como podemos ver más adelante en la visualización de la ACF, cada 12 meses siempre hay un pico y vemos que a medida que aumenta el lag disminuye la autocorrelación.

```
df %>%
  ACF(X.Passengers, lag_max = 48) %>%
  autoplot() + labs(title="ACF",x='Lag en Meses',y='ACF')
```



Estacionariedad (stacionarity) Una vez entendidos los anteriores conceptos, podemos explicar la **Estacionariedad** (más sencillo stacionarity, en inglés para evitar confusiones con la estacionalidad). Una serie temporal es estacionaria si las propiedades de la misma no depende del tiempo en que es observada. Es decir, que no es afectada ni por tendencia ni por estacionalidad. Para que una serie sea estacionaria debe satisfacer:

1. La media de la serie temporal debe ser constante a través del tiempo.
2. La varianza de la serie temporal debe ser constante a través del tiempo.
3. La autocorrelación no debe variar con el tiempo.

Si bien visualmente se ve que esto no sucede hay dos tests para poder comprobarlo. ADF y KPSS.

```
adf.test(df$X.Passengers,k = 0) #--> da 0.01
```

```
## Warning in adf.test(df$X.Passengers, k = 0): p-value smaller than printed p-
## value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$X.Passengers
## Dickey-Fuller = -4.6392, Lag order = 0, p-value = 0.01
## alternative hypothesis: stationary
```

```
kpss.test(df$X.Passengers)
```

```
## Warning in kpss.test(df$X.Passengers): p-value smaller than printed p-value
```

```
##
##  KPSS Test for Level Stationarity
##
```

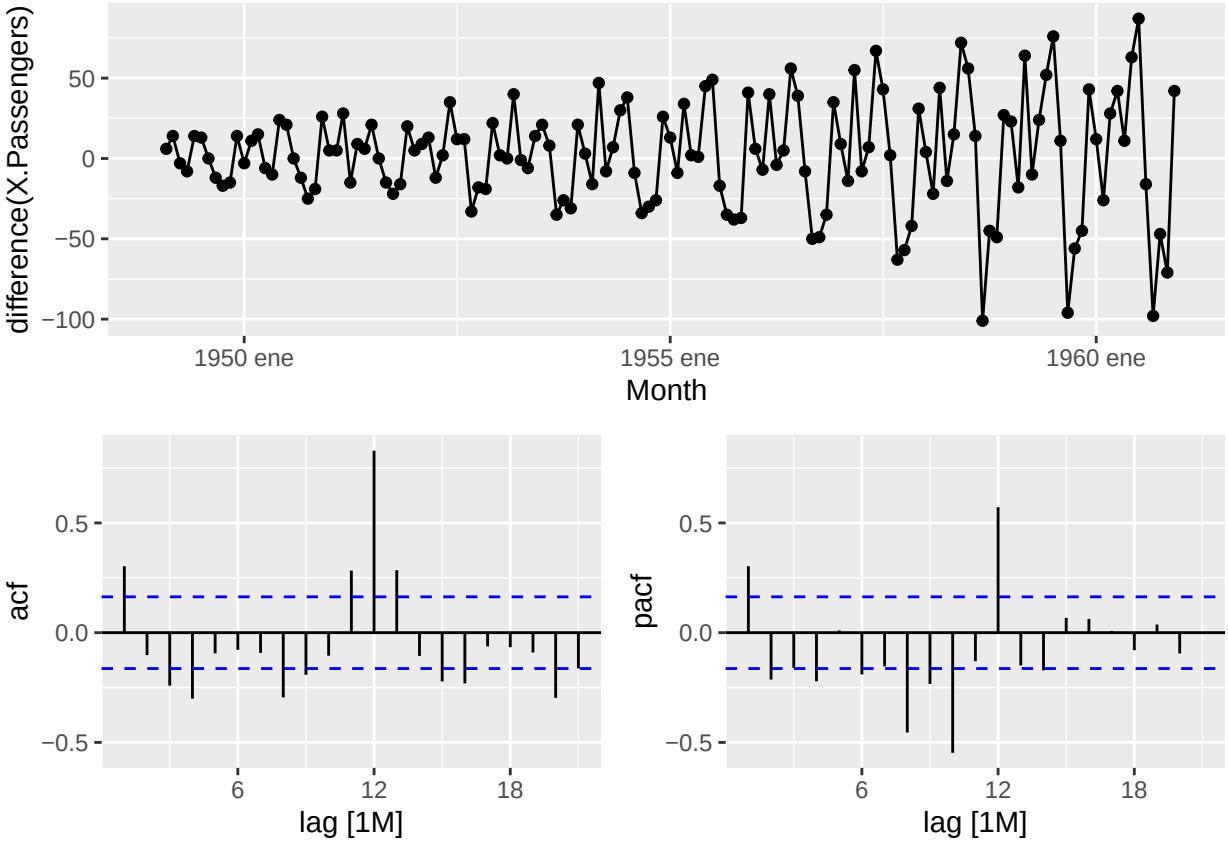
```
## data:  df$X.Passengers
## KPSS Level = 2.7395, Truncation lag parameter = 4, p-value = 0.01
```

Para sacar tendencia puedo restar el valor pasado. Veamos como queda visualmente

```
df_diff <- df %>% mutate(diff_passengers = difference(X.Passengers))

df %>%
  gg_tsdisplay(difference(X.Passengers), plot_type='partial')
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

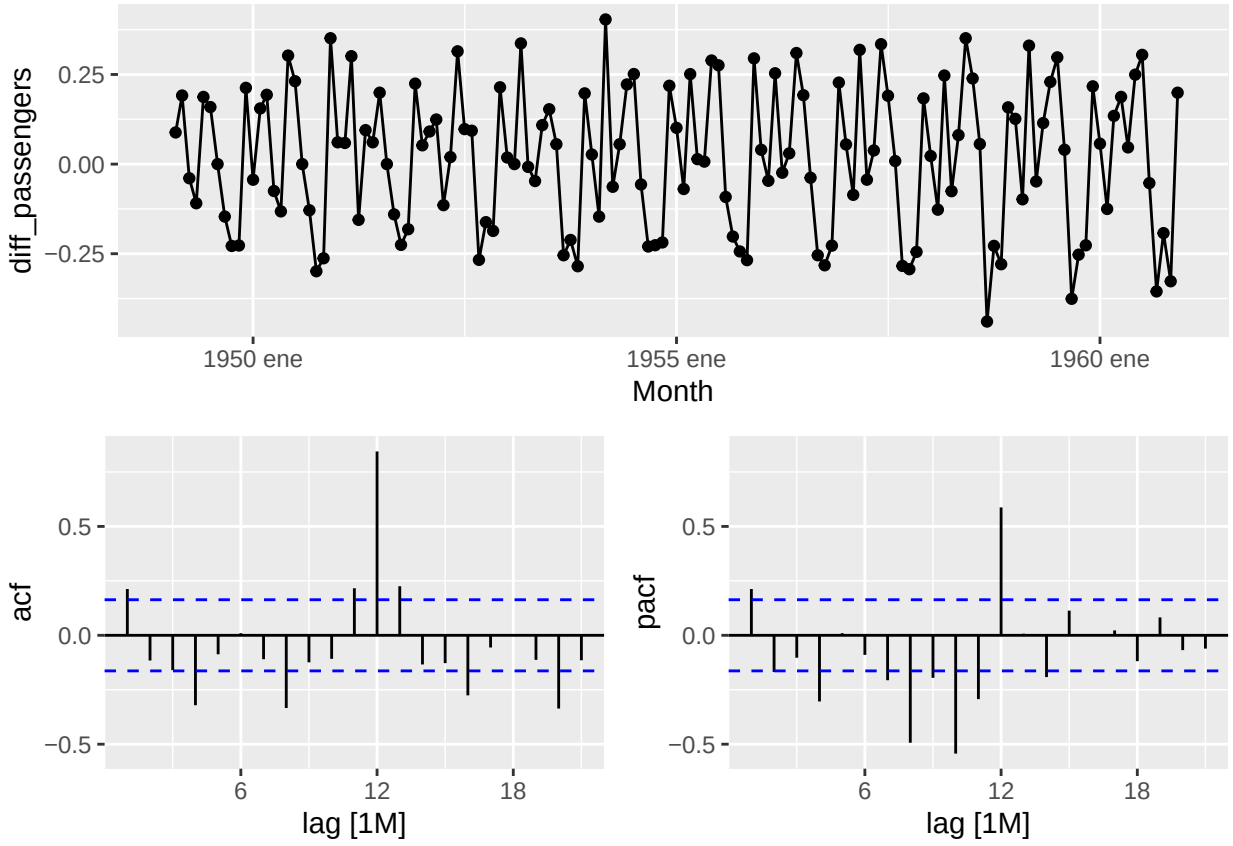


Claramente influye el problema que la varianza es mayor a medida que aumenta el tiempo, por lo que tengo que realizar una transformacion Box-Cox primero.

```
#df %>% autoplot(X.Passengers)
lambda <- BoxCox.lambda(df$X.Passengers)

#df%>%
#  autoplot(BoxCox(X.Passengers,lambda), colour = "black")


df_diff <- df %>% mutate(diff_passengers = difference(BoxCox(X.Passengers,lambda),1))

df_diff %>%
  gg_tsdisplay(diff_passengers, plot_type='partial')
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

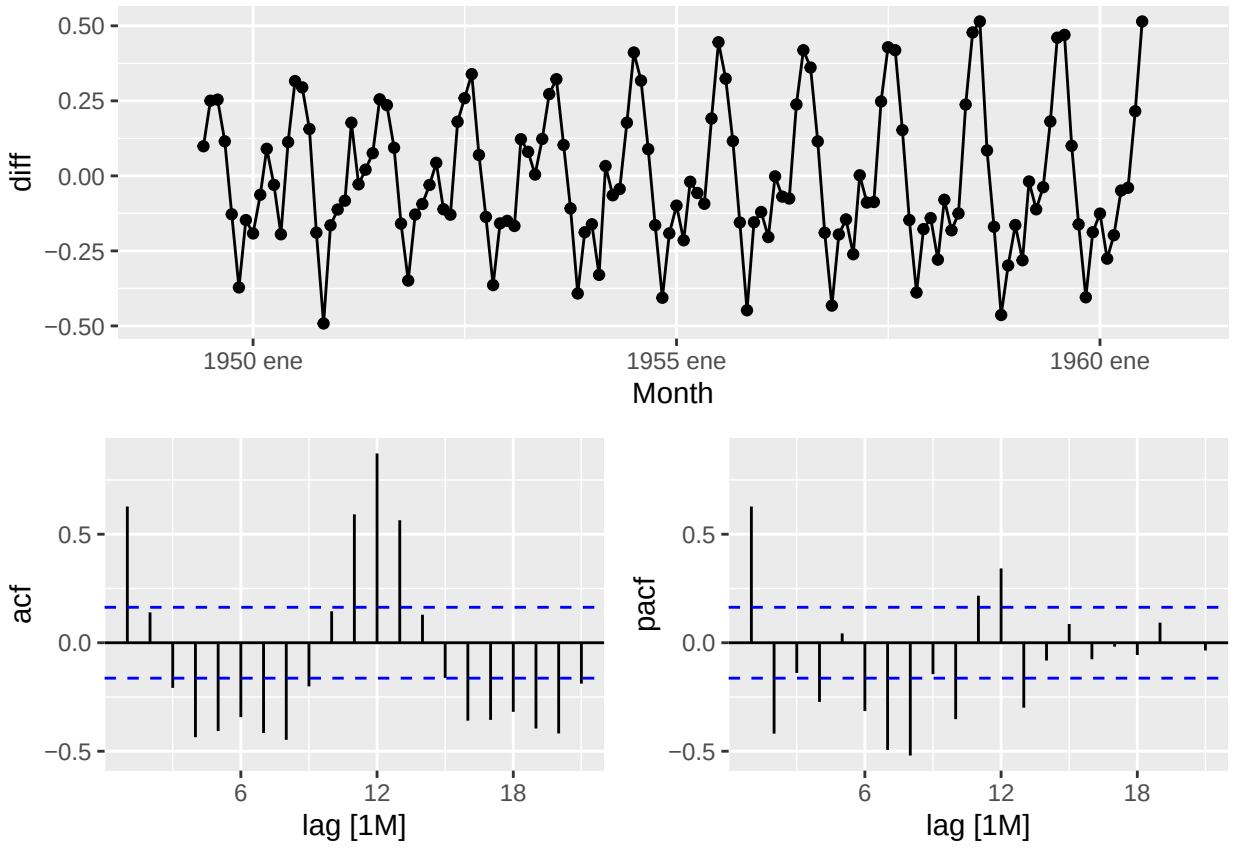


Y si en vez de restar el anterior uso una resta por media movil

```
df_movil_diff <- df %>%
    mutate(
    "tres" = slider::slide_dbl(X.Passengers, mean,
                .before = 5, .after = 5, .complete = TRUE)
  )

df_movil_diff <- df_movil_diff %>% mutate(diff = BoxCox(X.Passengers,lambda)-BoxCox(tres,lambda))

df_movil_diff %>%
  gg_tsdisplay(diff, plot_type='partial')
```

```
## Warning: Removed 10 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 10 rows containing missing values (geom_point).
```



Finalmente decido hacer boxcox y diferenciacion.

```
lambda <- BoxCox.lambda(df$X.Passengers)


df_diff <- df %>% mutate(boxcox_passengers = (BoxCox(X.Passengers,lambda)))


df_diff <- df_diff %>% mutate(diff_passengers = difference(BoxCox(X.Passengers,lambda)))


#Separo en Train y Test para los modelos

train_ts_ds_diff = df_diff %>% slice(-(133:144))
test_ts_ds_diff = df_diff %>% slice(133:144)

train_ts_ds = df %>% slice(-(133:144))
test_ts_ds = df %>% slice(133:144)

myts_diff <- ts(train_ts_ds_diff$diff_passengers, start=c(1949, 1), end=c(1959, 12), frequency=12)
```

Modelos

Modelo Auto Regresivo (AR)

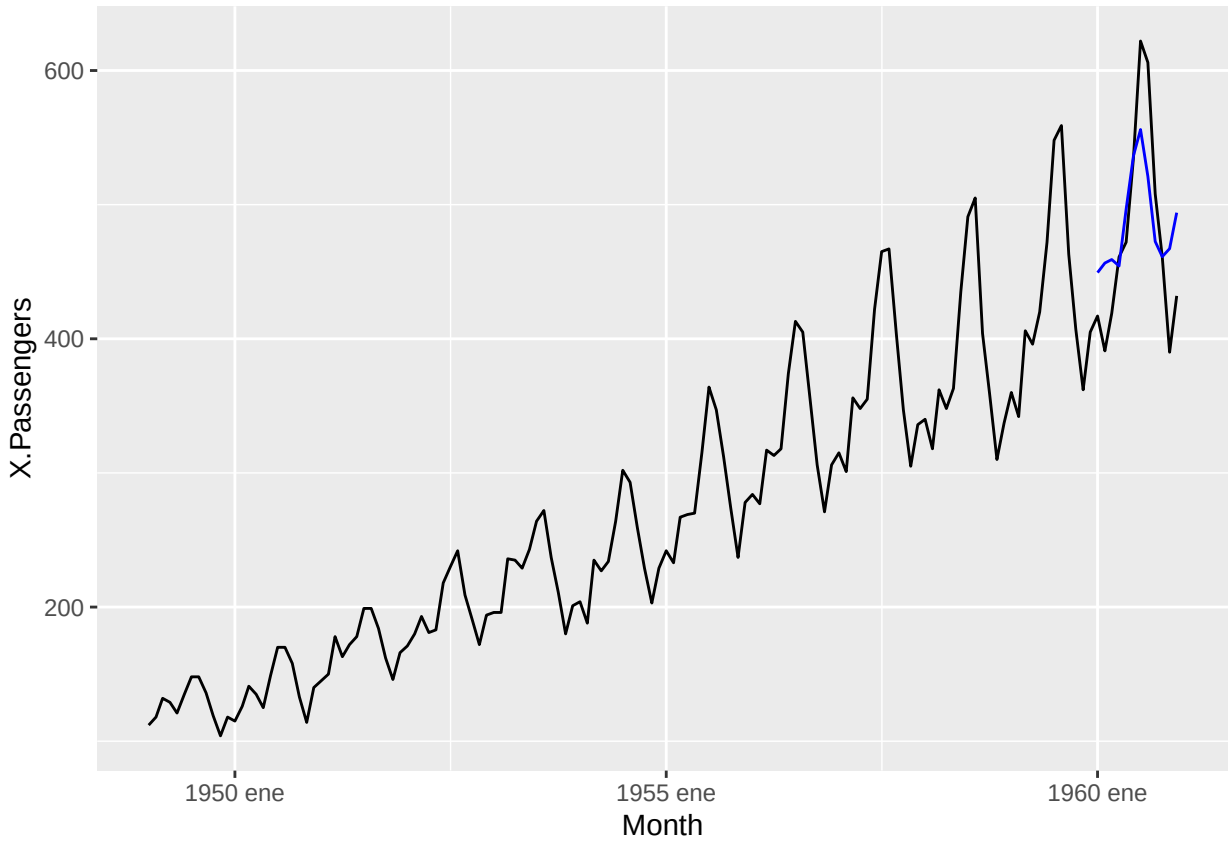
En este modelo se usa los valores pasados de la variable para predecir el siguiente. Tiene como parametro **p** que regula la cantidad de valores en el pasado que se consideran.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t \tag{3}$$

Los valores de los coeficientes ϕ_i seran cercanos a $+1$ si en y_{t-i} la serie copia al valor. Por el contrario, en caso que los coeficientes sean cercanos a -1 se implica que la serie se está autoregulando, por ejemplo valores de acciones de subas que se compensan con bajas.

Ejemplo de Modelo AR

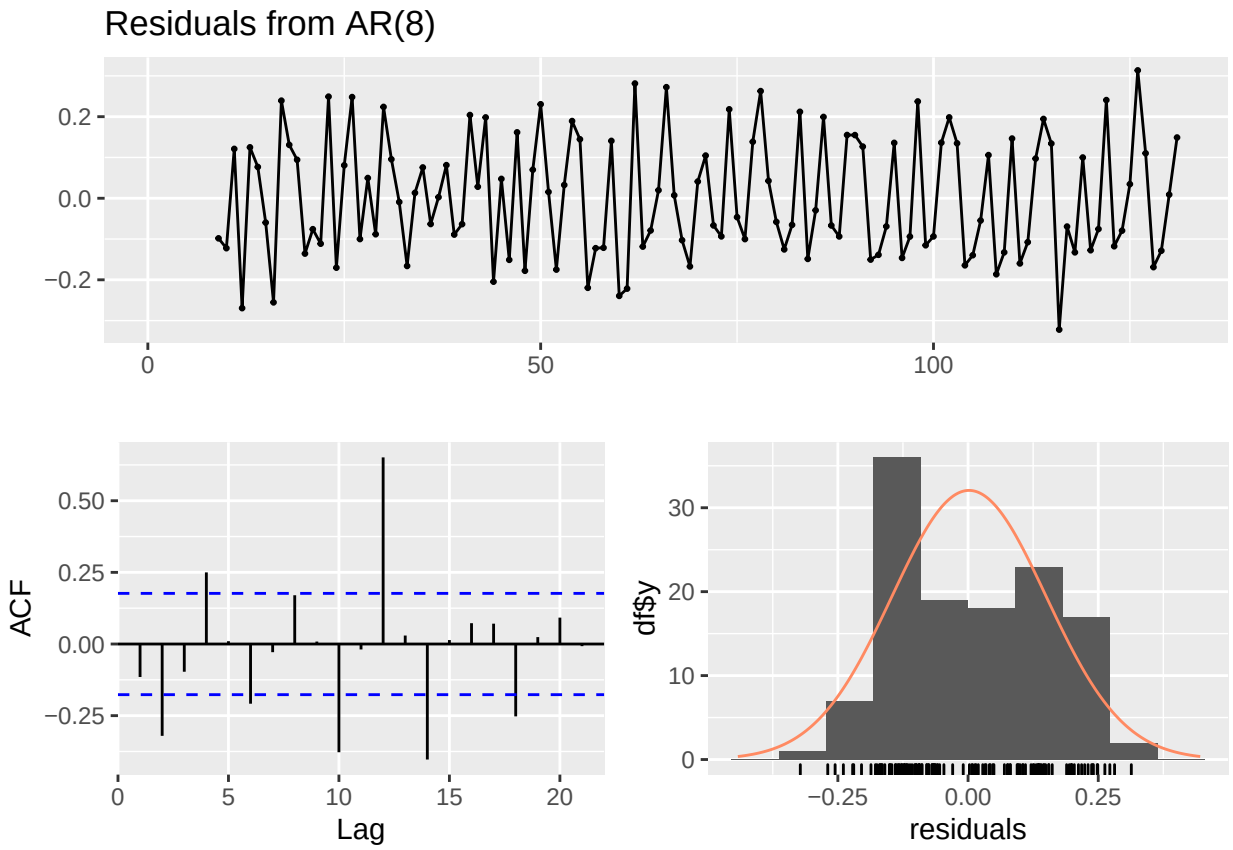
```
#AR (11,1,0) con diff
#fit_train_ar_210_diff = arima(x = (train_ts_ds_diff$diff_passengers),seasonal = F, order = c(11, 0, 0)
fit_train_ar_210_diff = ar(x = (train_ts_ds_diff$diff_passengers[-1]),order.max = 8,aic = FALSE)
predict_ar_210_diff <- predict(fit_train_ar_210_diff, n.ahead=12)


test_ts_ds_diff$prediccion_ar_210 = predict_ar_210_diff$pred

pred_diff_ar_210 = append(train_ts_ds_diff[132,]$diff_passengers,predict_ar_210_diff$pred)


#Grafico Transformado

#ggplot() + geom_line(data=df_diff, aes(x=Month,y=diff_passengers)) + geom_line(data=test_ts_ds_diff,ae


#Grafico invirtiendo todo
pred_inv_diff_ar_210 <-c()
pred_inv_diff_ar_210[1] <- train_ts_ds_diff$boxcox_passengers[132]
for(i in 2:length(pred_diff_ar_210)){
  pred_inv_diff_ar_210[i] <- pred_inv_diff_ar_210[i-1]+pred_diff_ar_210[i]
}
pred_transformed_ar_210 <- InvBoxCox(pred_inv_diff_ar_210,lambda)




test_ts_ds_diff$prediccion_transformada_ar_210 = pred_transformed_ar_210[2:13]



ggplot() + geom_line(data=df_diff, aes(x=Month,y=X.Passengers)) + geom_line(data=test_ts_ds_diff,aes(x=
```



Analisis Residuos

```
checkresiduals(fit_train_ar_210_diff)
```

```
## Warning in modeldf.default(object): Could not find appropriate degrees of
## freedom for this model.
```



Modelo Medias Moviles (MA)

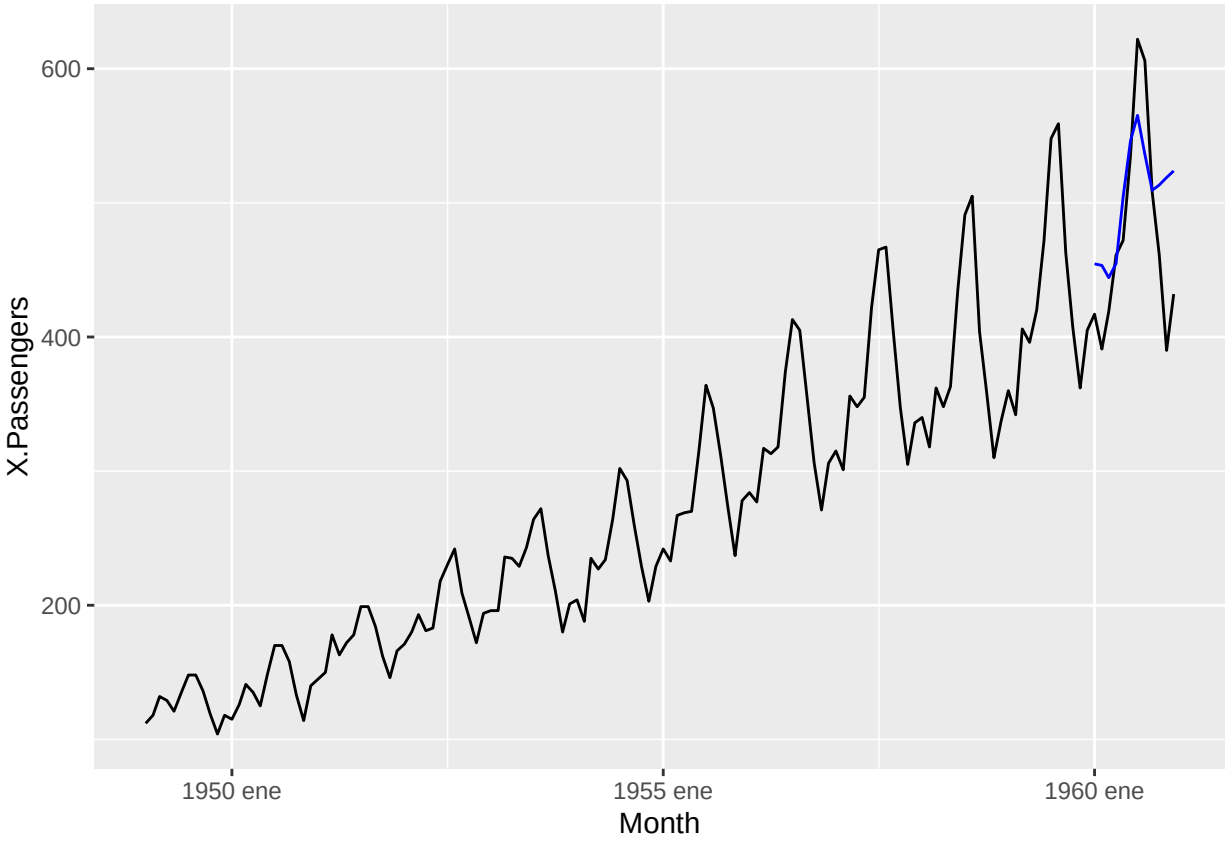
En el modelo de medias moviles se predice teniendo en cuenta los residuos pasados. El parametro **q** regula la cantidad de términos de errores que se consideran. Soluciona los problemas del modelo AR para captar cambios bruscos.

$$y_t = c + \epsilon_t + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t \tag{4}$$

Ejemplo de Modelo MA

```
#MA (0,1,11) con diff
#fit_train_ma_014_diff = ar(x = (train_ts_ds_diff$diff_passengers),order = c(0, 0, 11))
fit_train_ma_014_diff <- arima(x=(train_ts_ds_diff$diff_passengers), order=c(0,0,11), seasonal = list(o
predict_ma_014_diff <- predict(fit_train_ma_014_diff, n.ahead=12)


test_ts_ds_diff$prediccion_ma_014 = predict_ma_014_diff$pred

pred_diff_ma_014 = append(train_ts_ds_diff[132,]$diff_passengers,predict_ma_014_diff$pred)


#Grafico Transformado

#ggplot() + geom_line(data=df_diff, aes(x=Month,y=diff_passengers)) + geom_line(data=test_ts_ds_diff,ae
```

```r
#Grafico invirtiendo todo
pred_inv_diff_ma_014 <-c()
pred_inv_diff_ma_014[1] <- train_ts_ds_diff$boxcox_passengers[132]
for(i in 2:length(pred_diff_ma_014)){
  pred_inv_diff_ma_014[i] <- pred_inv_diff_ma_014[i-1]+pred_diff_ma_014[i]
}
pred_transformed_ma_014 <- InvBoxCox(pred_inv_diff_ma_014,lambda)



test_ts_ds_diff$prediccion_transformada_ma_014 = pred_transformed_ma_014[2:13]



ggplot() + geom_line(data=df_diff, aes(x=Month,y=X.Passengers)) + geom_line(data=test_ts_ds_diff,aes(x=
```



Modelo ARIMA

El modelo ARIMA (*Auto Regressive Integrated Moving Average*) combina ambos modelos.

$$y_t = c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + ... + \phi_p y'_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} + \epsilon_t \quad (5)$$

Depende de tres párametros **(p,d,q)**, siendo **p** y **q** los mencionados anteriormente para AR y MA respectivamente, y d que permite regular el shift de diferenciación. Permite captar lo mejor de ambos modelos ya que permite compensar de errores anteriores (MA) y a su vez, mantener la similaridad de valores anteriores (AR).

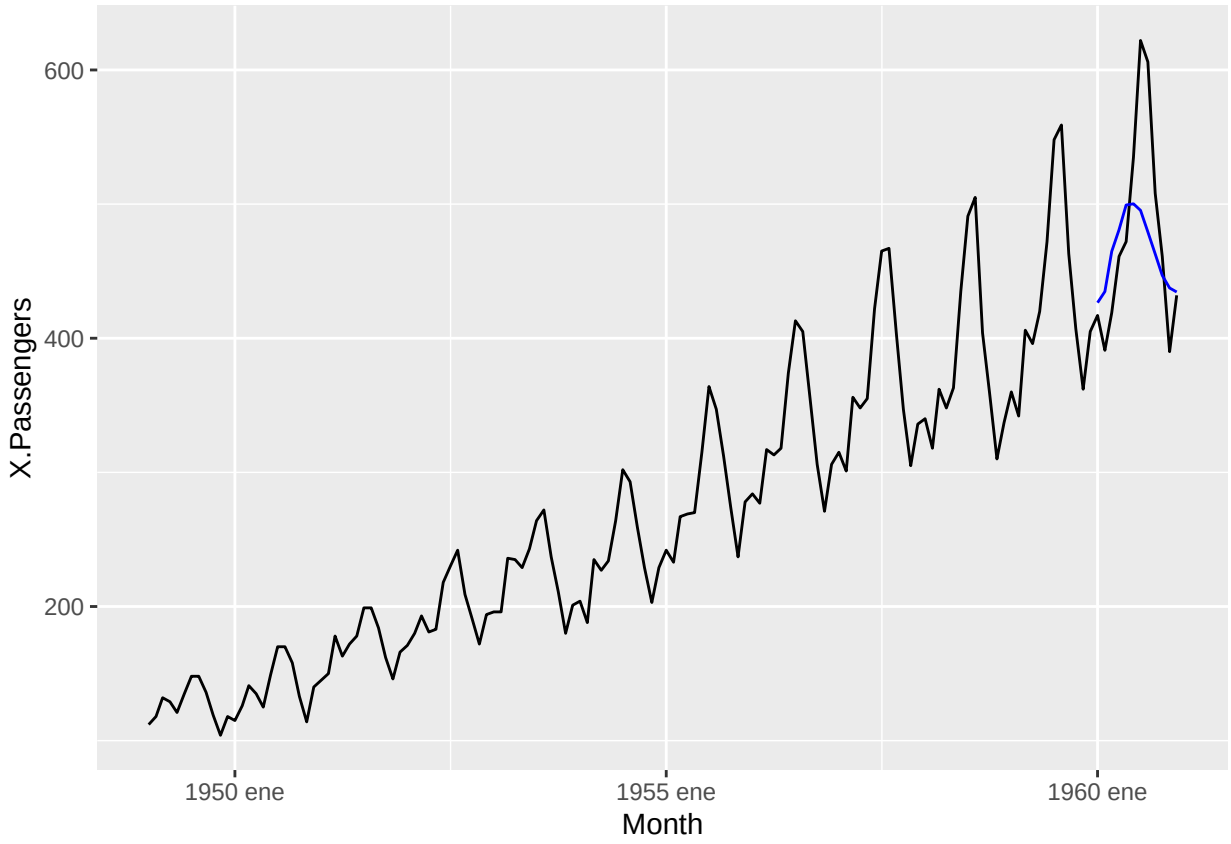
Ejemplo ARIMA

```
fit_am_diff = auto.arima(myts_diff, seasonal =F, ic ="aic", trace=TRUE)
```

```
##
##  ARIMA(2,0,2)            with non-zero mean : Inf
##  ARIMA(0,0,0)            with non-zero mean : -51.26116
##  ARIMA(1,0,0)            with non-zero mean : -54.6228
##  ARIMA(0,0,1)            with non-zero mean : -56.57301
##  ARIMA(0,0,0)            with zero mean     : -52.1867
##  ARIMA(1,0,1)            with non-zero mean : -61.45544
##  ARIMA(2,0,1)            with non-zero mean : Inf
##  ARIMA(1,0,2)            with non-zero mean : Inf
##  ARIMA(0,0,2)            with non-zero mean : Inf
##  ARIMA(2,0,0)            with non-zero mean : -56.61692
##  ARIMA(1,0,1)            with zero mean     : -62.51988
##  ARIMA(0,0,1)            with zero mean     : -57.82346
##  ARIMA(1,0,0)            with zero mean     : -55.86587
##  ARIMA(2,0,1)            with zero mean     : -69.45624
##  ARIMA(2,0,0)            with zero mean     : -57.58737
##  ARIMA(3,0,1)            with zero mean     : -68.37673
##  ARIMA(2,0,2)            with zero mean     : -66.67393
##  ARIMA(1,0,2)            with zero mean     : -66.86094
##  ARIMA(3,0,0)            with zero mean     : -56.9632
##  ARIMA(3,0,2)            with zero mean     : -70.62148
##  ARIMA(4,0,2)            with zero mean     : -74.72335
##  ARIMA(4,0,1)            with zero mean     : -66.4325
##  ARIMA(5,0,2)            with zero mean     : -72.72427
##  ARIMA(4,0,3)            with zero mean     : Inf
##  ARIMA(3,0,3)            with zero mean     : -98.43886
##  ARIMA(2,0,3)            with zero mean     : -67.13975
##  ARIMA(3,0,4)            with zero mean     : Inf
##  ARIMA(2,0,4)            with zero mean     : -87.66485
##  ARIMA(4,0,4)            with zero mean     : Inf
##  ARIMA(3,0,3)            with non-zero mean : Inf
##
##  Best model: ARIMA(3,0,3)            with zero mean
```

```
forecast_diff<-predict(fit_am_diff, n.ahead=12)
pred_con_ult_train_diff = append(train_ts_ds_diff[132,]$diff_passengers,forecast_diff$pred)


#Grafico Transformado
test_ts_ds_diff$prediccion <- pred_con_ult_train_diff[-1]

#ggplot() + geom_line(data=df_diff, aes(x=Month,y=diff_passengers)) + geom_line(data=test_ts_ds_diff,ae


#Grafico invirtiendo todo
pred_inv_diff <-c()
pred_inv_diff[1] <- train_ts_ds_diff$boxcox_passengers[132]
for(i in 2:length(pred_con_ult_train_diff)){
  pred_inv_diff[i] <- pred_inv_diff[i-1]+pred_con_ult_train_diff[i]
}
```

```
pred_transformed <- InvBoxCox(pred_inv_diff,lambda)



test_ts_ds_diff$prediccion_transformada = pred_transformed[2:13]



ggplot() + geom_line(data=df_diff, aes(x=Month,y=X.Passengers)) + geom_line(data=test_ts_ds_diff,aes(x=
```



Modelo S-ARIMA (Seasonal ARIMA)

S-ARIMA permite complementar el modelo ARIMA agregando análisis estacional. En este caso además de los parámetros **(p,d,q)** no estacionales, se le agregan **(P,D,Q)_m** estacionales. Para este ya no es necesario realizar las transformaciones anteriores ya que los parametros pueden captar esta información.

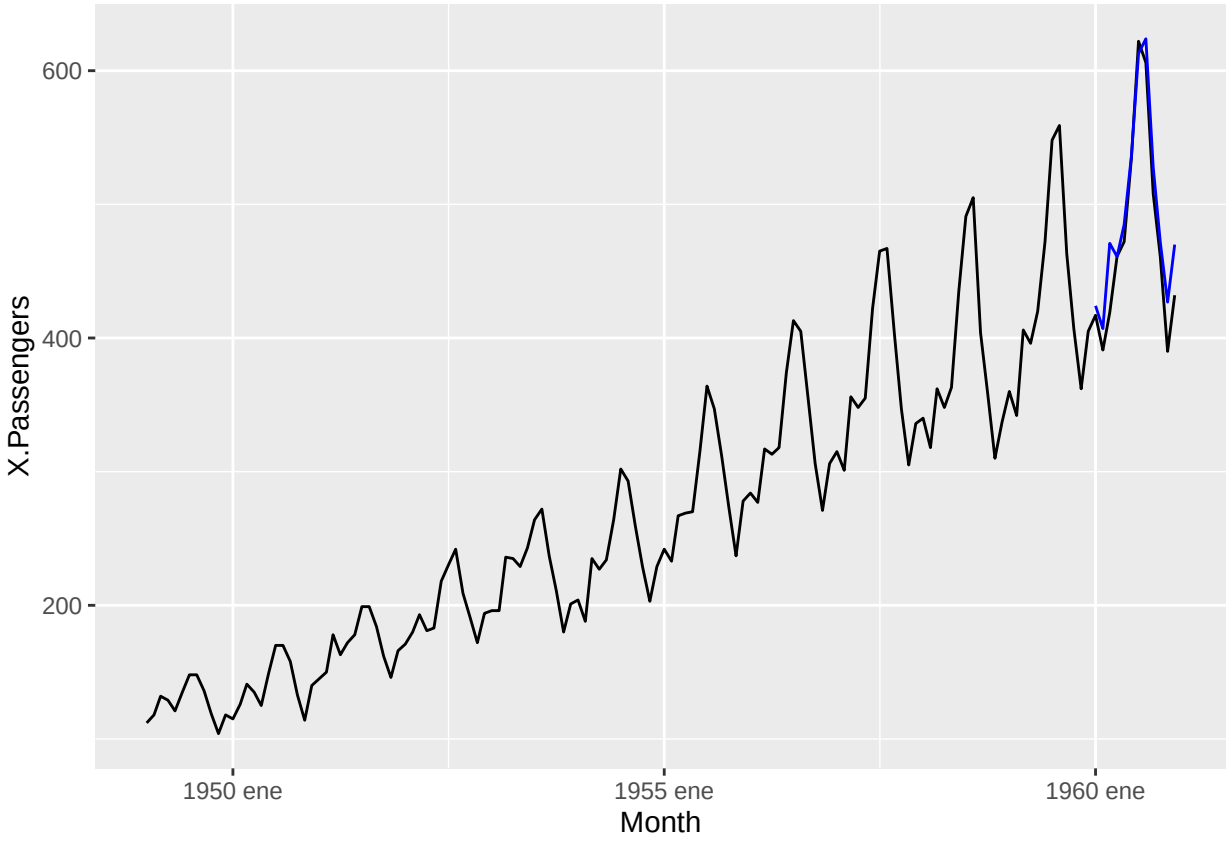
Ejemplo S-ARIMA

```
#fiteo con autoarima sin dif ni log para que capte el seasonal

myts <- ts(train_ts_ds$X.Passengers, start=c(1949, 1), end=c(1959, 12), frequency=12)
fit_am = auto.arima(myts, seasonal = T, ic ="aic", trace=TRUE)
```

```
##
##  ARIMA(2,1,2)(1,1,1)[12]                    : Inf
##  ARIMA(0,1,0)(0,1,0)[12]                    : 905.0652
```

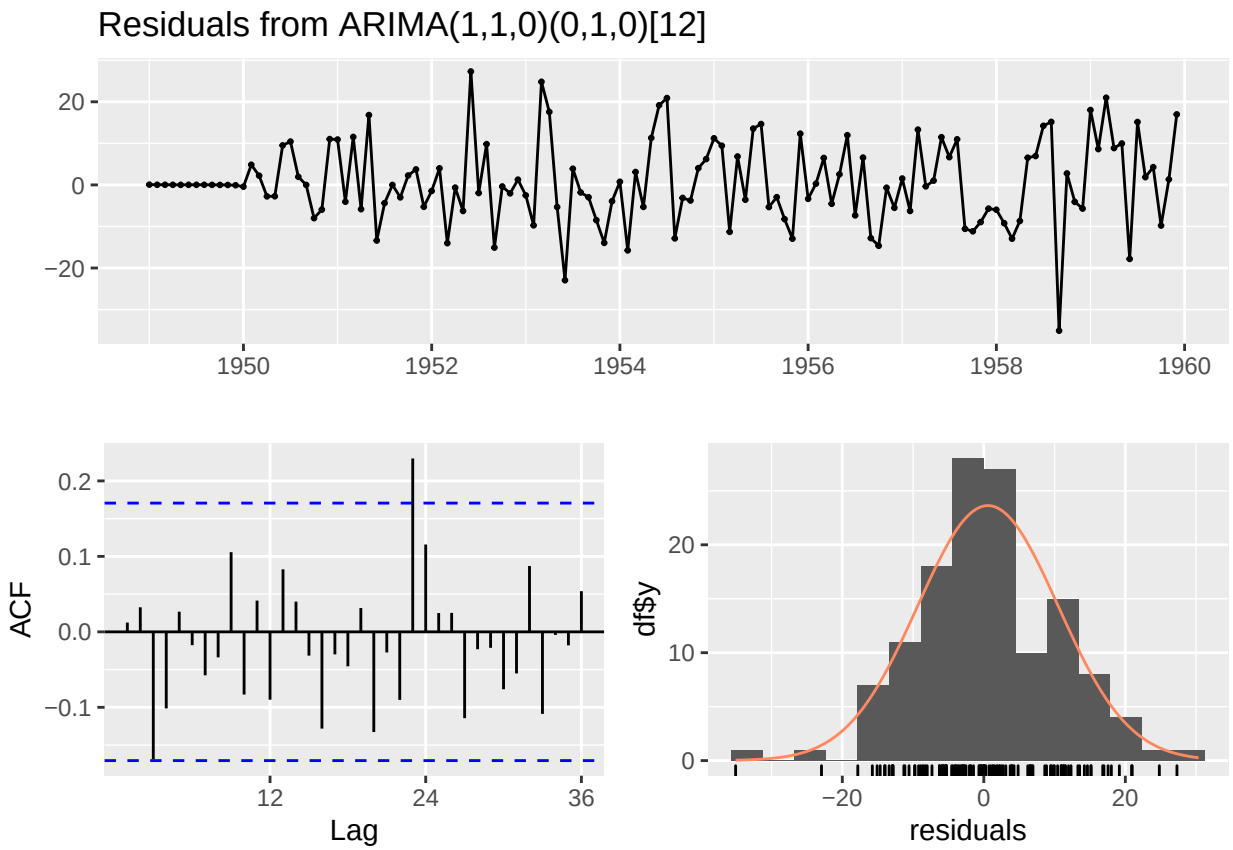
```
##  ARIMA(1,1,0)(1,1,0)[12]                    : 900.8231
##  ARIMA(0,1,1)(0,1,1)[12]                    : 901.7211
##  ARIMA(1,1,0)(0,1,0)[12]                    : 899.9021
##  ARIMA(1,1,0)(0,1,1)[12]                    : 901.0524
##  ARIMA(1,1,0)(1,1,1)[12]                    : Inf
##  ARIMA(2,1,0)(0,1,0)[12]                    : 901.3376
##  ARIMA(1,1,1)(0,1,0)[12]                    : 900.9716
##  ARIMA(0,1,1)(0,1,0)[12]                    : 900.6852
##  ARIMA(2,1,1)(0,1,0)[12]                    : 902.9668
##
##  Best model: ARIMA(1,1,0)(0,1,0)[12]
```

```
#fit_train_autoarima = auto.arima(ts(train_ts_ds$Passengers), seasonal = T, ic ="aic", trace=TRUE)
forecast_3<-forecast(fit_am, h=12, level=c(95))


predict_3<-predict(fit_am, n.ahead=12)

pred_con_ult_train_3 = append(train_ts_ds[132,]$X.Passengers,predict_3$pred)



test_ts_ds$prediccion <- pred_con_ult_train_3[-1]

ggplot() + geom_line(data=df, aes(x=Month,y=X.Passengers)) + geom_line(data=test_ts_ds,aes(x=Month,y=pr
```



Analisis Residuos S-ARIMA

```
checkresiduals(fit_am)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,0)(0,1,0)[12]
## Q* = 29.817, df = 23, p-value = 0.1547
##
## Model df: 1.   Total lags used: 24
```

Referencias