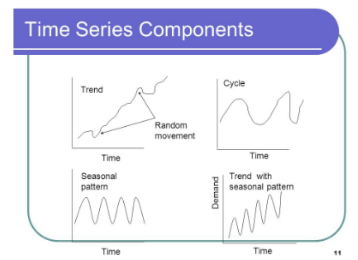
**Series de Tiempo II**

Vamos a trabajar con la **componente cíclica** de las series de tiempo. Una serie de tiempo puede descomponerse en Tendencia, Estacionalidad, Ciclo y error aleatorio:



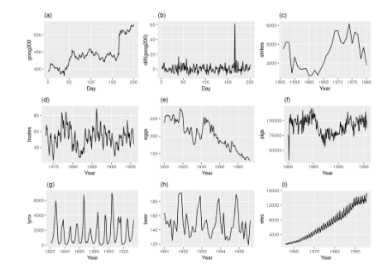


El **ciclo** es cualquier tipo de dinámica no capturada por la tendencia o la estacionalidad. Cuando vemos algún tipo de dinámica que vincule el presente con el pasado estamos ante un ciclo. No necesariamente debe tratarse de un ciclo rígido. Es más complejo analizar ciclos que analizar tendencia o estacionalidad.

**Series Estacionarias**: Podemos trabajar con ciclos sólo si tenemos **series estacionarias.** Las series estacionarias son aquellas en las cuales las propiedades estadísticas del proceso que las genera no cambian en el tiempo, o tengan mínimas variaciones. Entonces podemos analizarlas y predecir las series de tiempo a partir de estas propiedades.

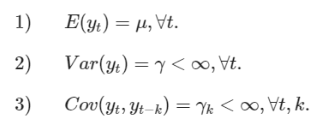


Las series con tendencia o estacionalidad no son estacionarias.



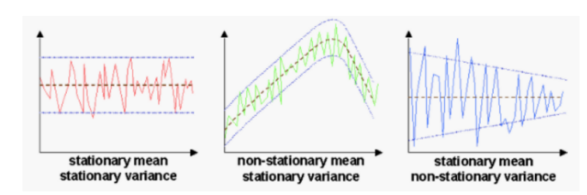
d, h e i tienen estacionalidad; a, c, e, f, i tienen tendencia; b y g son series estacionarias. Aunque g pareciera ser no estacionario, lo es. El problema es que como los ciclos no se repiten a intervalos regulares, son impredecibles.

Una serie es estacionaria (o estacionaria en sentido débil) si:



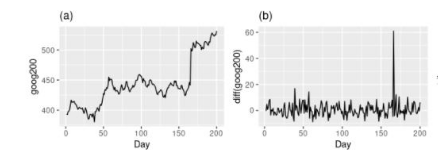
1. El promedio es constante a lo largo del tiempo.
2. La varianza es un valor constante y finito a lo largo del tiempo.
3. La autocovarianza es un valor constante. Esto es la covarianza de la variable contra sí misma en otro período de tiempo. Debe depender sólo de la tralsación k (períodos que me muevo); no del tiempo.

Acá tenemos ejemplos de series donde se representa el valor constante de las medidas; se puede observar la estacionariedad de la serie:

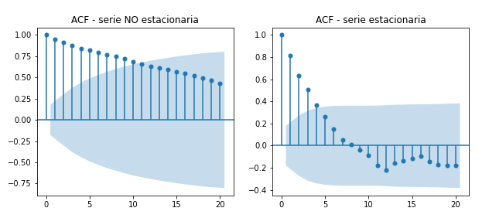


Hay maneras de transformar series no estacionarias en series estacionarias. Una es computar las diferencias entre las observaciones separadas por un lag, generalmente de 1 (consecutivas). A este proceso se lo llama **Diferenciación (**differencing**)**. Cuenta con un **orden de integración**, que indica el número de veces que debe ser aplicada en forma recursiva para transformar a una serie en estacionaria. Generalmente es de orden 1 (se aplica una sola vez).

Acá podemos ver a la izquierdala serie del valor de las acciones de Google y a la derecha las diferencias entre días consecutivos:



La **autocorrelación (ACF)** es la correlación de la variable contra sí misma en otro momento en el tiempo. La podemos usar para verificar si la serie es o no estacionaria: Si calculamos ACF aumentando el lag entre ambas series, en las series no estacionarias, ACF va a descender más lentamente; mientras que en las series estacionarias, el descenso a cero será más rápido:



**En Python:**

Vamos a usar una serie de precios de las acciones de Intel. Entre 2010 y 2020 mensualizada. Vamos a crear nuevas variables para aplicar un modelo de Transformación Logarítmica + Estacionalidad mensual:

df = pd.read\_csv(‘../Data/INTC.csv’, sep = ‘/t’, parse\_dates = [‘date’], index\_col = ‘date’)

df[‘timeIndex’] = pd.Series(np.arange(len(df[‘value’])), index = df.index)

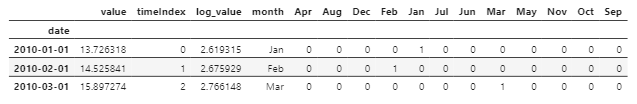
df[‘log\_value’] = np.log(df[‘value’])

df[‘month’] = [d.strftime(‘%b’) for d in df.index]

dummies\_mes = pd.get\_dummies(df[‘month’])

df = df.join(dummies\_mes)

df.head(3)



Vamos a crear el modelo sobre train y predecir el valor (logaritmo de la serie ‘model\_log\_est’). Luego generaremos el back transformation ’back\_model\_log\_est’ con una función exponencial:

from sklearn.model\_selection import train\_test\_split

df\_train, df\_test = train\_test\_split(df, test\_size = 12, random\_state = 42, shuffle = False)

model\_log\_est = smf.ols(‘log\_value ~ timeIndex + Aug + Dec + Feb + Jan + Jul + Jun + Mar + May + Nov + Oct + Sep’, data = df\_train).fit()

df\_train[‘model\_log\_est’] = model\_log\_est.predict(df\_train[[‘timeIndex’, ‘Aug’, ‘Dec’, ‘Feb’, ‘Jan’, ‘Jul’, ‘Juln’, ‘Mar’, ‘May’, ‘Nov’, ‘Oct’, ‘Sep’]])

df\_test[‘model\_log\_est’] = model\_log\_est.predict(df\_test[[‘timeIndex’, ‘Aug’, ‘Dec’, ‘Feb’, ‘Jan’, ‘Jul’, ‘Juln’, ‘Mar’, ‘May’, ‘Nov’, ‘Oct’, ‘Sep’]])

df\_train[‘back\_model\_log\_est’] = np.exp(df\_train[‘model\_log\_est’])

df\_test[‘back\_model\_log\_est’] = np.exp(df\_test[‘model\_log\_est’])

Vamos a verificar analíticamente la performance del modelo:

def RMSE(predicted, actual):

mse = (predicted - actual) \*\* 2

rmse = np.sqrt(mse.sum()/mse.count())

return rmse

df\_Results = pd.DataFrame(columns = [‘Model’, ‘RMSE’])

df\_Results.loc[0, ‘Model’] = ‘Log Model + est’

df\_Results.loc[1, ‘RMSE’] = RMSE(df\_test[‘back\_model\_log\_est’], df\_test[‘value’])

df\_Results



Una manera de detector estacionalidad es a través de Tests estadísticos de Hipótesis de Estacionaridad, tales como el **test de Dickey-Fuller aumentado (ADF)**.

Se trata de un test en el cual la hipótesis nula H0 indica que la serie no es estacionaria; es decir, que si p > 0.05, H0 no se rechaza y se considera que la serie no es estacionaria; caso contrario, se rechaza la hipótesis nula y la serie es estacionaria.

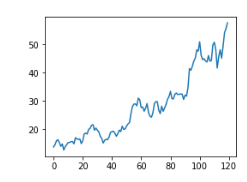
Vamos a aplicar este test a los valores originales de la serie:

**En Python:**

plt.figure(figsize=(4,3))

plt.plot(df\_train.timeIndex, df\_train.value)

plt.show()



Con la función **adfuller** podemos implementar ADF. Recibiremos como resultados:

* el **p-value**, con el cual, si es mayor a 5%, no rechazaremos la hipótesis nula y consideraremos que la serie es no estacionaria.
* El valor del **estadístico**, el cual si es menor a determinados valores críticos consideraremos la serie como estacionaria.
* **Valores críticos**.

**En Python:**

from statsmodels.tsa.stattools import adfuller

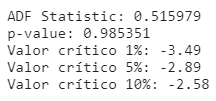
result = adfuller(df\_train[‘value’])

print(‘ADF Statistic: %f’ % result[0])

print(‘p-value %f’ % result[1])

for key in result[4].items():

print(‘Valor crítico %s: %.2f’ % (key, value))



Como el p-value es mayor a 0.05, no se rechaza la hipótesis nula y se considera que la serie es no estacionaria.

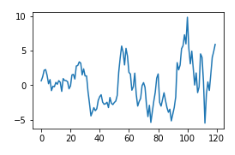
Ahora vamos a aplicar ADF al residuo de la serie entre los valores originales y el valor predicho con back transformation:

res\_model = df\_train[‘value’] – df\_train[‘back\_model\_log\_est’]

plt.figure(figsize = (4, 2.5))

plt.plot(df\_train.timeIndex, res\_model, ‘-‘)

plt.show()



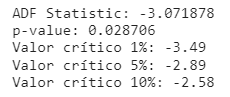
result = adfuller(res\_model)

print(‘ADF Statistic: %f’ % result[0])

print(‘p-value: %f’ % result[1])

for key, value in result[4].items():

print(‘Valor crítico %s: %.2f’ % (key,value))



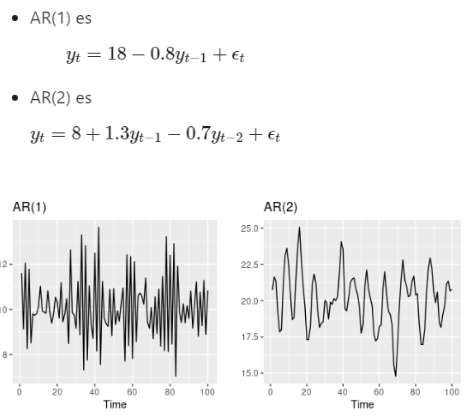
En este caso, como el p-value es menor a 0.05, se rechaza la hipótesis nula y se considera la serie como estacionaria.

**Modelos AR(p) y MA(q):**

**Modelo Autorregresivo AR(p)**: Autorregresión es una regresión de la variable contra si misma. Se pronostica la variable a partir de una combinación lineal de valores pasados de la serie. Se trata de un modelo autorregresivo de orden p que se expresa con la siguiente fórmula:



Siendo εt ruido blanco y los yt-x son los p valores anteriores de la serie. Manejan un amplio rango de patrones de series de tiempo en función de cuántos parámetros usamos:



Si usamos AR(1), hay casos particulares:

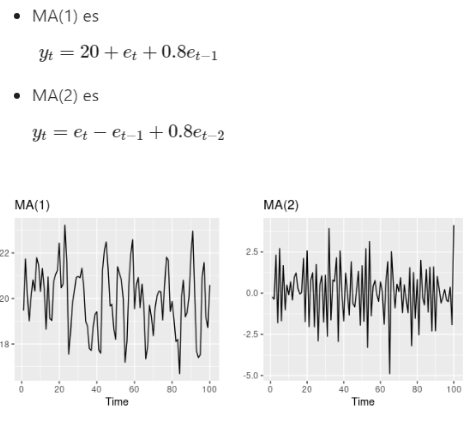
* Si φ1 = 0 entonces tenemos sólo ruido blanco. 
* Si φ1 = 1 y c = 0, entonces  Tenemos un modelo de **Random Walk**.
* Si φ1 = 1 y c != 0 entonces  Tenemos un modelo de **Random Walk con deriva**.
* Será estacionario en aquellos casos donde |φ1| < 1

**Modelo de Media Movil MA(q)**: Usa shocks inobservables presentes y pasados (ruido blanco) basados en un modelo similar a una regresión. Se trata de un modelo de media móvil de orden q que se expresa con la siguiente fórmula:



Siendo et ruido blanco o rezagos. q entonces son la cantidad de rezagos considerados en el modelo. Esto es, la parte de la serie que no puede explicarse ni con tendencia, ni con estacionalidad.

Análogamente a los modelos autorregresivos, los modelos de media móvil también pueden manejar un amplio rango de patrones de series de tiempo:



**Importante!:** MA(q) **no es** el modelo de Media Movil (Moving Average o Rolling Mean) visto en Series de Tiempo I.

MA(q) se usa para pronosticar valores futuros. La Media Movil se usa para estimar el ciclo de la tendencia en los valores pasados.

**Modelo ARIMA (AutoRegressive Integrated Moving Average):**

El modelo ARIMA sin estacionalidad surge de la combinación de un modelo Auto-Regresivo AR(p), con un modelo de media móvil MA(q) y un proceso de diferenciación con un orden de integración.

**ARIMA (p, d, q):**

El número de términos AR p es el orden p de la parte del modelo AR(p).

El número de términos MA q es el orden q de la parte del modelo MA(q).

El número de diferenciaciones d es el orden de integración del proceso de diferenciación. Se se trata de una serie estacionaria, este proceso no es necesario y d = 0.

Podemos determinar el valor de p y q:

* Haciendo un **plot de la Función de Autocorrelación (ACF)**: Entonces vamos a ver las autocorrelaciones que miden la relación entre yt y yt-k, para diferentes valores de k.
* Haciendo un **plot de la Función de Autocorrelación Parcial (PACF)**: De esta manera vamos a ver la misma relación entre yt y yt-k, pero eliminando los efectos que generan las correlaciones con los valores intermedios yt-x, con x entre 1 y k-1.

**En Python:**

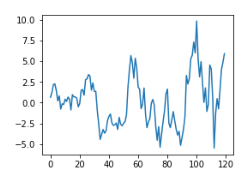
Vamos a trabajar con los residuos de la última serie calculada (diferencias entre el valor original de la serie y los valores predichos por el modelo con back transformation).

res\_model = df\_train[‘value’] – df\_train[‘back\_model\_log\_est’]

plt.figure(figsize=(4,3))

plt.plot(df\_train.timeIndex, res\_model, ‘-’)

plt.show()



Vamos a aplicarle el test de Dickey-Fuller ADF para comprobar si es o no estacionaria:

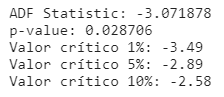
result = adfuller(res\_model)

print(‘ADF Statistic: %f’ % result[0])

print(‘p-value: %f’ % result[1])

for key, value in result[4].items():

print(‘Valor crítico %s: %2.f’ % (key, value))



Como el p-value es menor a 0.05, rechazamos la hipótesis nula y llegamos a la conclusión de que la serie es estacionaria.

Para determinar los valores de p y q, necesitamos plotear ACF y PACF. En un mundo ideal, lo que queremos es que no haya ninguna correlación entre la serie y los rezagos de la misma. Todos los picos deberían caer dentro de la región azul en los gráficos. Estos plots nos pueden servir para determinar modelos **ARIMA(p,d,0)** o bien **ARIMA (0, d, q)**.

La serie seguirá un modelo **ARIMA(p,d,0)** si:

* ACF cae en forma exponencial o sinusoide.
* PACF tiene un pico importante en el lag p y ningún otro luego de p; caen dentro de la región azul.

La serie seguirá un modelo **ARIMA(0,d,q)** si:

* PACF cae en forma exponencial o sinusoide.
* ACF tiene un pico importante en el lag q y ningún otro luego de q.

Esta regla puede complementarse con otra en donde el cero se reemplaza con los primeros rezagos significativamente mayores a cero.

**En Python:**

from statsmodel.tsa.stattools import acf, pacf

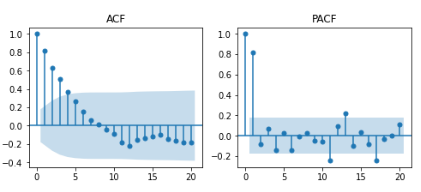
from statsmodels.graphics.staplots import plot\_acf, plot\_pacf

fig, axes = plt.subplots(1, 2, figsize=(8,3))

smt.graphics.plot\_acf(res\_model, lags=20, ax = axes[0], title=’ACF’)

smt.graphics.plot\_pacf(res\_model, lags=20, ax = axes[1], title=’PACF’)

plt.show()



ACF cae en forma exponencial y PACF tiene dos picos significativos. Entonces podemos asumir p = 2, y q = 0; o bien también podríamos asumir p = 2 y q = 2.

Ahora vamos a instanciar el modelo con los parámetros determinados en el análisis de ACF y PACF: p = 2; d = 0 (poque se trata de una serie estacionaria) y q = 0.

**En Python:**

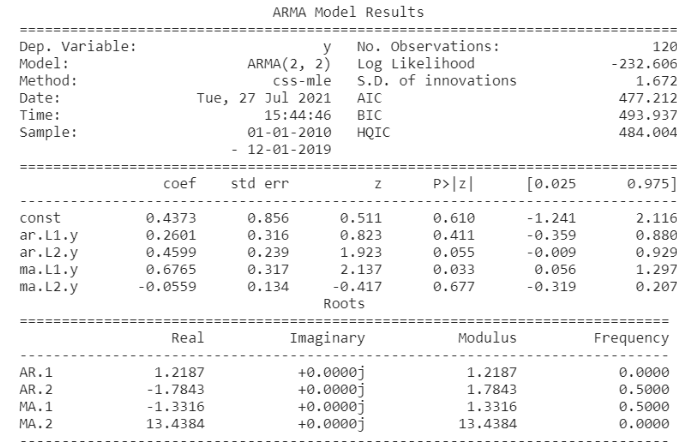
from statsmodels.tsa.arima\_model import ARIMA

model\_ARIMA = ARIMA(res\_model, order=(2,0,2))

results\_ARIMA = mode\_ARIMA.fit()

Vamos a analizar los resultados. Algunos de ellos son **AIC** (Criterio de Información de Akaike) y **BIC** (Criterio de Información Baysesiano (Schwartz)); los cuales cuanto más pequeños son, mejor es el modelo.

print(results\_ARIMA.summary())



Con el método **resid()** podemos analizar los residuos del modelo ARIMA. En este caso, parecen estar correctos, con media 0 y varianza uniforme.

**En Python:**

residuals = pd.DataFrame(results\_ARIMA.resid)

fig, axes = plt.subplots(2, 2, figsize=(10, 4))

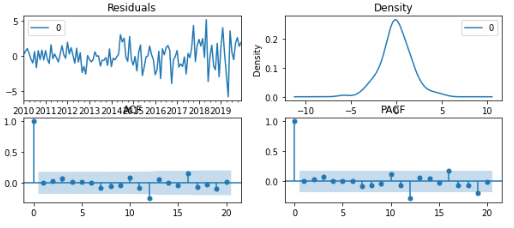
residuals.plot(title=’Residuals’, ax = axes[0,0])

residuals.plot(kind=’kde’, title=’Density’, ax = axes[0,1])

sm.graphics.plot\_acf(residuals, lags = 20, ax = axes[1,0], title = ‘ACF’)

sm.graphics.plot\_pacf(residuals, lags = 20, ax = axes[1,1], title = ‘PACF’)

plot.show()



Observando los plots de ACF y PACF, todos los picos caen dentro de la región azul, y esto nos indica que no existe autocorrelación. El residuo entonces es ruido blanco.

Vamos a realizar las predicciones del modelo con el método **forecast()**.

**En Python:**

predictions\_ARIMA, se, conf = results\_ARIMA.forecast(len(df\_test[‘value’]), alpha = 0.05)

Se suman los resultados del modelo ARIMA a los valores originales. El modelo se hizo sobre los residuos de la última serie calculada: diferencias entre valor original de la serie y los valores predichos por el modelo con back transformation.

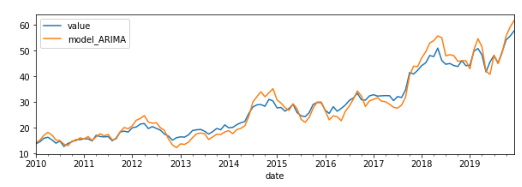
**En Python:**

df\_train[‘model\_ARIMA’] = df\_train[‘value’] + results.ARIMA.fittedvalues

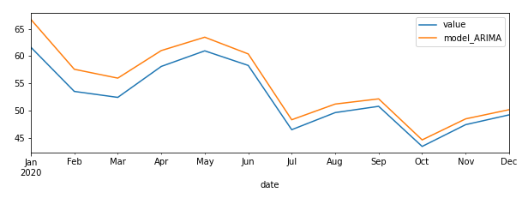
df\_test[‘model\_ARIMA’] = df\_test[‘value’] + predictions\_ARIMA

Vamos a comparar la serie con las estimaciones del modelo, en train y test:

df\_train.plot(kind = ‘line’, y = [‘value’, ‘model\_ARIMA’], figsize=(10,3))



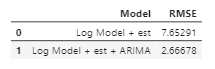
df\_test.plot(kind = ‘line’, y = [‘value’, ‘model\_ARIMA’], figsize=(10,3))



Ahora vamos a calcular analíticamente la mejora que presenta ARIMA respecto al modelo anterior:

df\_Results.loc[1, ‘Model’] = ‘Log Model + es + ARIMA’

df\_Results.loc[1, ‘RMSE’] = RMSE(df\_test[‘model\_ARIMA’], df\_test[‘value’])



El resultado mejora un montón.

**Conclusiones:**

* Estudiamos series estacionarias, cómo evaluar si una serie es o no estacionaria y cómo transformar una serie no estacionaria en una serie estacionaria.
* Vimos el Modelo **ARIMA** el cual es una combinación de un modelo Auto-Regresivo (AR(p)) + un modelo de Media Movil MA(q) + un proceso de diferenciación (d).
* Vimos cómo determinar los parámetros p y q para el modelo ARIMA.
* Analizamos resultados del modelo ARIMA en un ejemplo concreto.
* Con ARIMA vimos que podemos profundizar su estudio para desarrollar mejores predicciones.