ARIMA - Python

September 6, 2022

Juan Isaula

1 Introducción a las series temporales y la estacionariedad

Bienvenido a este curso sobre pronósticos utilizando modelos ARIMA en Python. Mi nombre es Juan Isaula y seré su guia miestras aprende a predecir el futuro de las series temporales.

1.0.1 Motivación

Los datos de series temporales estan en todas partes en este mundo. Se utiliza en una amplia variedad de campos. Hay muchos conjuntos de datos para los que nos gustaría poder predecir el futuro. Conocer el futuro de las tasas de obesidad podría ayudarnos a intervenir ahora por la salud pública; predecir las demandas de energía de los consumidores podría ayudar a que las centrales eléctricas funcionen de manera más eficiente; y predecir cómo cambiará la población de una ciudad podría ayudarnos a construir la infraestructura que necesitaremos.

1.0.2 Contenido del curso

Podemos pronosticar todos estos conjuntos de datos utilizando modelos de series temporales, y los modelos ARIMA son una de las herramientas de serie temporal a las que acudir. Aprenderá cómo ajustar estos modelos y cómo optimizarlos. Aprenderá cómo hacer pronósticos de datos importantes del mundo real y, lo que es más importante, cómo encontrar los límites de sus pronósticos.

1.0.3 Carga y trazado

Comencemos examinando una serie de tiempo. Podemos cargar una serie temporal desde csv usando pandas. Aquí establecemos el índice como la columna de fecha (**index_col**) y analizamos la fecha en el tipo de datos de fecha y hora (**parse dates**).

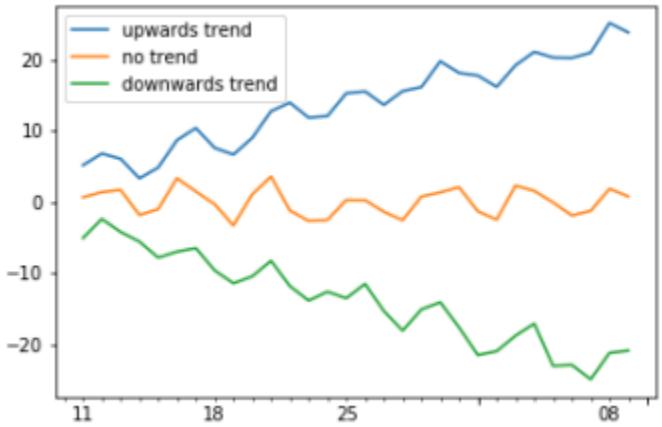
[3]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("/Users/juanisaula/Downloads/candy_production.csv",index_col =␣
 ↪"date",parse_dates = True)
df.head()
```

[3]:

date	IPG3113N
1972-01-01	85.598809
1972-02-01	71.741404
1972-03-01	65.948809
1972-04-01	64.490724
1972-05-01	64.933842

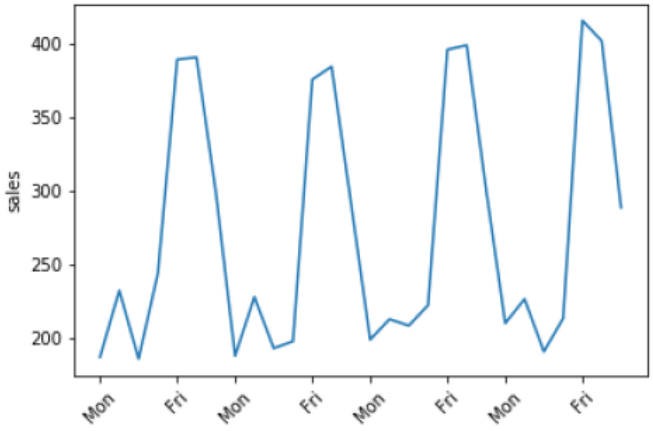
1.0.4 Tendencia

Para graficar los datos, hacemos una figura de pyplot y usamos el método de diagrama de puntos de DataFrame. Una característica importante de una serie de tiempo es su tendencia. Una tendencia positiva es una línea que generalmente se inclina hacia arriba: los valores aumentan con el tiempo. Del mismo modo, una tendencia negativa es donde los valores disminuyen



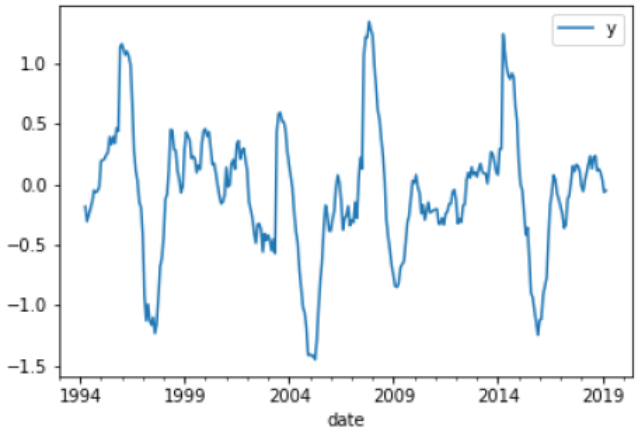
1.0.5 Estacionalidad

Otra característica importante es la estacionalidad. Una serie temporal estacional tiene patrones que se repiten a intervalos regulares, por ejemplo, ventas altas todos los fines de semana.



1.0.6 Ciclicidad

Por el contrario, la ciclicidad es donde hay un patrón repetitivo pero no un perído fijo.



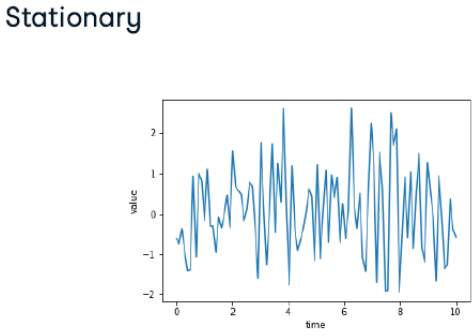
1.0.7 Rido Blanco

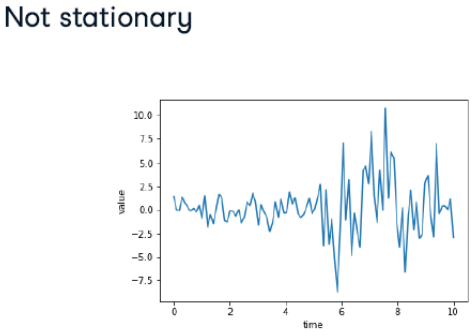
El ruido blanco es un concepto importante en series temporales y modelos ARIMA. El ruido blanco es una serie de mediciones, donde cada valor no está correlacionado con los valores anteriores. Puede pensar en esto como lanzar una moneda, el resultado de un lanzamiento de moneda no depende de los resultados de los lanzamientos de moneda anteriores. De manera similar, con el ruido blanco, el valor de la serie no depende de los valores anteriores.

1.0.8 Estacionariedad

Para modelar una serie de tiempo, debe ser estacionaria. Estacionario significa que la distribución de los datos no cambia con el tiempo. Para que una serie temporal sea estacionaria debe cumplir tres criterios. Estos son:

- La serie tiene tendencia cero, no crece ni se reduce.
- La varianza es constante.
- La distancia promedio de los puntos de datos desde la linea cero no cambia.





1.0.9 Division de datos de prueba y entrenamiento

En general, en el aprendizaje automático, tiene un conjunto de entrenamiento en el que ajusta su modelo y un conjunto de prueba con el que probará sus predicciones. La previsión de series de tiempo es exactamente lo mismo. Sin embargo, nuestra división de prueba de entrenamiento (training) será diferente. Usamos los valores pasados para hacer predicciones futuras, por lo que necesitaremos dividir los datos en el tiempo. Entrenamos con los datos anteriores en la serie temporal y probamos con los datos que vienen después. Podemos dividir series de tiempo en una fecha dada como se muestra a continuación:

[5]:
```
# Datos de Entrenamiento: Todos los datos hasta finales de 2018
df_train = df.loc[:"2018"]

# Datos de prueba: Todos los datos desde 2019 en adelante
df_test = df.loc["2019":]
```

Exploración

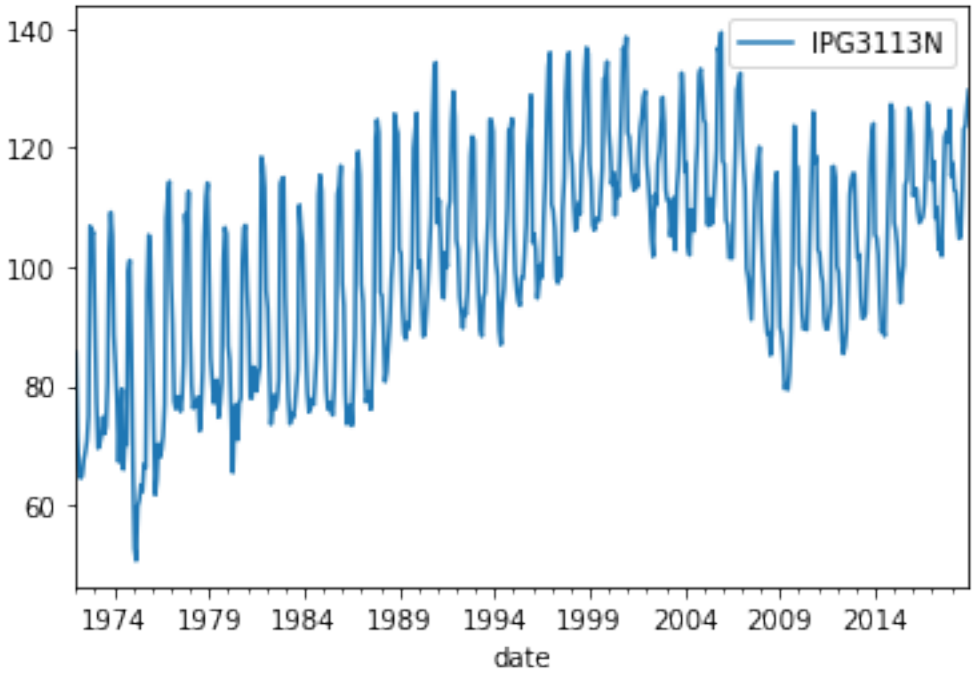
Puede hacer gráficos regularmente, pero en este curso, es importante que pueda controlar explícitamente en qué eje se trazan las diferentes series de tiempo. Esto será importante para que pueda evaluar sus predicciones de series temporales más adelante.

Aquí su tarea es trazar un conjunto de datos de la producción mensual de dulces de EE. UU. entre 1972 y 2018.

Específicamente, está trazando el índice de producción industrial IPG3113N. Esta es la cantidad total de azúcar y productos de confitería producidos en los EE. UU. por mes, como porcentaje de la producción de enero de 2012. Entonces 120 sería el 120% de la producción industrial de enero de 2012.

Vea cómo ha cambiado esta cantidad con el tiempo y cómo cambia a lo largo del año

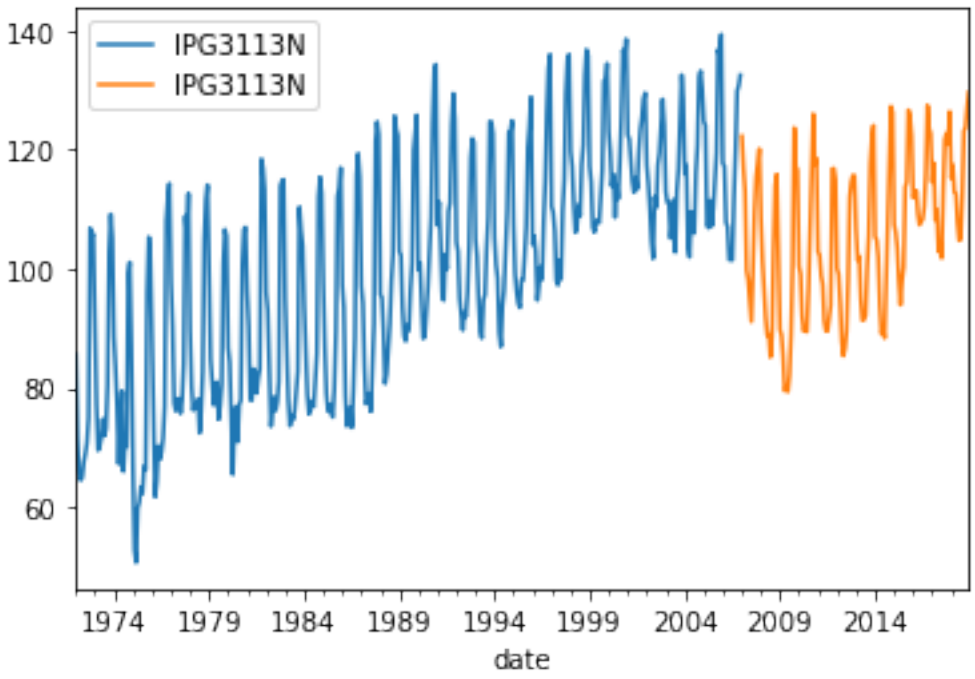
[6]:
```
# Importamos las librerias a utilizar
import matplotlib.pyplot as plt
import pandas as pd

# Cargamos la serie temporal de producción de dulces
caramelos = pd.read_csv('/Users/juanisaula/Downloads/candy_production.csv',
            index_col ='date',
            parse_dates =True)

# Trazamos y mostramos la serie de tiempo en el eje ax1
fig, ax1 = plt.subplots()
caramelos.plot(ax=ax1)
plt.show()
```



Divisiones de prueba y entrenamiento

En este ejercicio, tomará el conjunto de datos de producción de dulces y lo dividirá en un tren y un conjunto de prueba. La razón para hacer esto es para que puedas probar la calidad del ajuste de tu modelo cuando hayas terminado.

[7]:
```
# Dividimos los datos en un tren y un conjunto de prueba
caramelos_train = caramelos.loc[:'2006']
caramelos_test = caramelos.loc['2007':]

# Creaamos un eje ax
fig, ax = plt.subplots()

# Trazamos los datos de train y test en ax
caramelos_train.plot(ax=ax)
caramelos_test.plot(ax=ax)
plt.show()
```



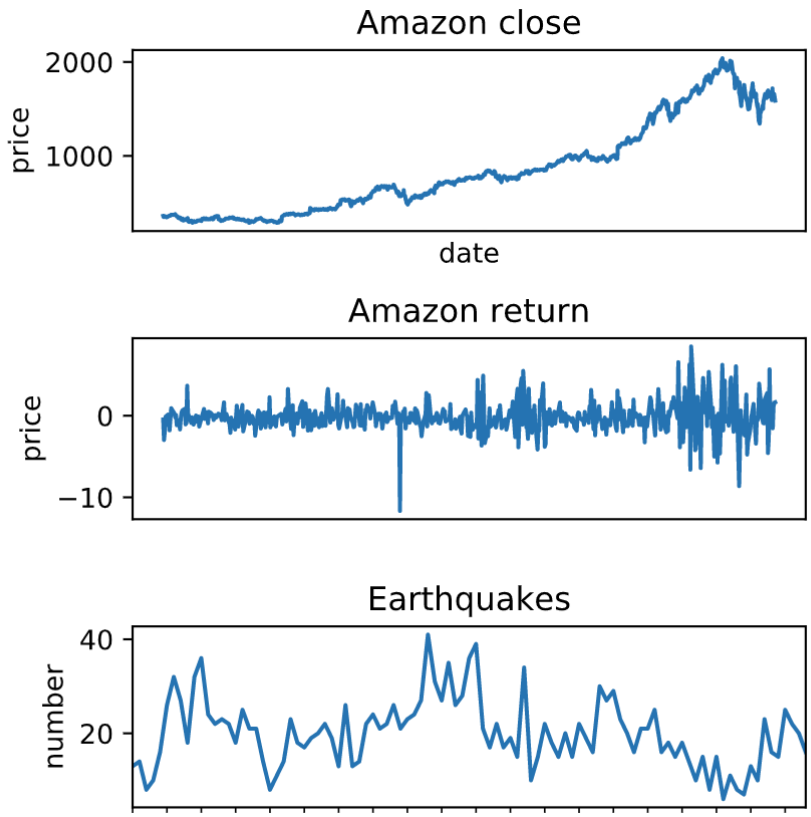
Es estacionaria

dentificar si una serie de tiempo es estacionaria o no estacionaria es muy importante. Si es estacionario, puede usar modelos ARMA para predecir los próximos valores de la serie temporal. Si no es estacionario, no puede usar modelos ARMA; sin embargo, como verá en la siguiente lección, a menudo puede transformar series temporales no estacionarias en estacionarias.

El gráfico que se muestra a continuación, presenta la serie temporal del precio de cierre de las acciones de amazon. ¿ El Precio de las acciones es estacionario? La respuesta es que No, porque el gráfico superior tiene una tendencia.

El gráfico central que se muestra es una serie temporal del rendimiento (aumento porcentual del precio por día) de las acciones de Amazon. ¿ El rendimiento de las acciones es estacionario? La respuesta es NO, porque en la gráfica del medio, la varianza cambia con el tiempo.

El gráfico inferior es una serie temporal del numero de grandes terremotos por año (terrenmoto de magnitud 7.0 o mayor). ¿Es estacionario el número de grandes terremotos por año? La respuesta es SI, la trama inferior parece estar estacionaria.



1.1 Hacer series de tiempo estacionarias

Ya hemos aprendido sobre las formas en que una serie de tiempo puede ser no estacionaria y cómo podemos identificarla al graficarla.

Sin embargo, existen formas más formales de realizar esta tarea, con pruebas estadísticas. También hay formas de transformar series temporales no estacionarias en estacionarias. Abordaremos ambos en esta lección y luego estará listo para comenzar a modelar.

1.1.1 Prueba de Dicky-Fuller Aumentada

La prueba más común para identificar si una serie de tiempo no es estacionaria es la prueba de Dicky-Fuller aumentada. Esta es una prueba estadística, donde la hipótesis nula es que su serie de tiempo no es estacionaria debido a la tendencia.

1.1.2 Aplicar la prueba de adfuller

Podemos implementar la prueba de Dicky-Fuller aumentada utilizando modelos estadísticos. Primero importamos la función adfuller como se muestra, luego podemos ejecutarla en nuestra serie temporal.

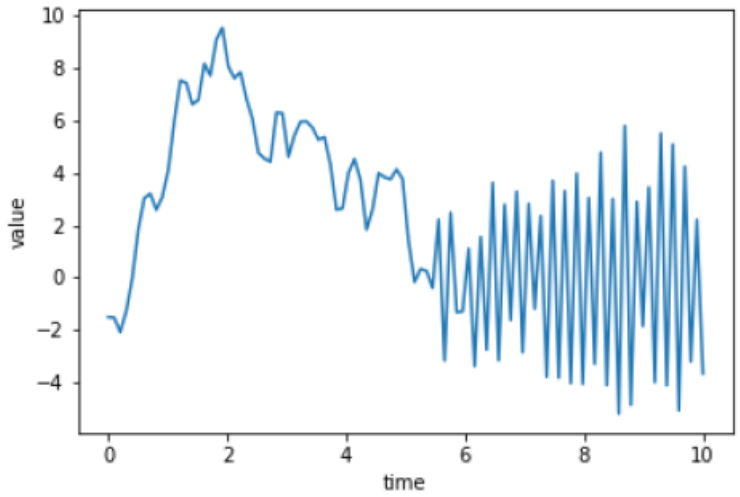
[8]:
```
from statsmodels.tsa.stattools import adfuller
df = pd.read_csv("/Users/juanisaula/Downloads/amazon_close.csv",index_col =␣
 ↪"date", parse_dates = True)
results = adfuller(df["close"])
```

```
print(results)
```

```
(-1.344669096532598, 0.6084966311408413, 23, 1235, {'1%': -3.4356560275160835,
'5%': -2.8638831211270817, '10%': -2.568017509711682}, 10782.877783880944)
```

El objeto de resultados es una tupla. - Elemento cero es el estadistico de prueba (-1.34). Cuanto más negativo sea este número, más probable es que los datos estén estacionarios. - El siguiente valor de la tupla es el valor p de la prueba, aquí es 0.60. Si es valor p < 0.05, rechazamos la hipótesis nula y asumimos que nuestra serie de tiempo debe ser estacionaria. - El último elemento de la tupla es un diccionario, este almacena los valores criticos del estadístico de prueba que equivalen a diferentes valores p. En este caso, si queríamos un valor de p de 0.05 o inferior, nuestra estadística de prueba debía estar por debajo de -2.91

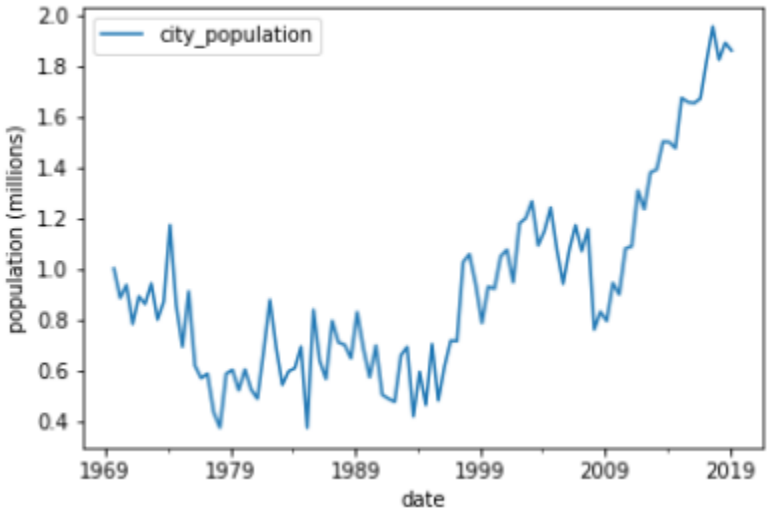
NOTA: Dicky-Fuller solo prueba la estacionariedad de tendencia. En este ejemplo, aunque el comportamiento de la serie temporal cambia claramente y no es estacionario, pasa la prueba de Dicky-Fuller.



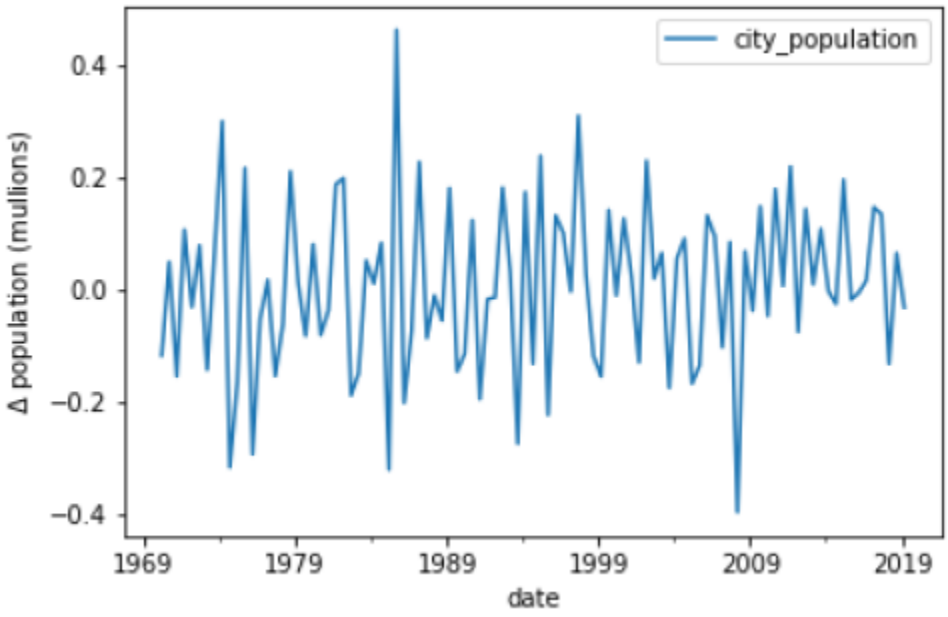
Tomando la Diferencia Así que digamos que tenemos una serie de tiempo que no es estacionaria. Necesitamos transformar los datos en una forma estacionaria antes de poder modelarlos. Puede pensar en esto un poco como la ingeniería de funciones en el aprendizaje automático clásico.

Comencemos con un conjunto de datos no estacionario. Aquí hay un ejemplo de la población de una ciudad. Una forma muy común de hacer que una serie de tiempo sea estacionaria es tomar su diferencia. Aquí es donde, de cada valor de nuestra serie temporal restamos el valor anterior. Es decir

$$\triangle y_t = y_t - y_{t-1}$$



Aqui esta la serie de tiempo despues de la diferenciación. Esta vez, tomar la diferencia fue suficiente para hacerla estacionaria, pero para otras series de tiempo es posible que necesitemos tomar la diferencia más de una vez.



Otras Transformaciones

A veces necesitaremos realizar otras transformaciones para que la serie temporal sea estacionaria. Esto podría ser tomar el logaritmo, o la raíz cuadrada de una serie de tiempo, o calcular el cambio porcentual. Puede ser dificil decidir cuál de estos hacer, pero a menudo la solución mas simple es la mejor.

Dicky-Fuller aumentado

En este ejercicio ejecutará la prueba de Dicky-Fuller aumentada en la serie temporal de terremotos para probar la estacionariedad. Trazó esta serie de tiempo en el último ejercicio. Parecía que

podría estar estacionario, pero los terremotos son muy dañinos. Si quieres hacer predicciones sobre ellos será mejor que estés seguro.

Recuerda que si no fuera estacionario esto significaría que el número de sismos por año tiene una tendencia y está cambiando. Esta sería una noticia terrible si tiene una tendencia al alza, ya que significa más daño. ¡También sería una noticia terrible si tuviera una tendencia a la baja, podría sugerir que el núcleo de nuestro planeta está cambiando y esto podría tener muchos efectos secundarios para nosotros!

[9]:
```
# Importar función de prueba de Dicky-Fuller aumentada
from statsmodels.tsa.stattools import adfuller

earthquake = pd.read_csv("/Users/juanisaula/Downloads/earthquakes.csv")

 # Corremos el test
result = adfuller(earthquake["earthquakes_per_year"])

# Estadistico del test
print(result[0])

# p-value
print(result[1])

# Valores criticos
print(result[4])
```

```
-3.183192251191782
0.02097842525600371
{'1%': -3.5003788874873405, '5%': -2.8921519665075235, '10%':
-2.5830997960069446}
```

La diferenciación debe ser la primera transformación que intente hacer que una serie de tiempo sea estacionaria. Pero a veces no es la mejor opción.

Una forma clásica de transformar series de tiempo de existencias es el logaritmo de la serie. Esto se calcula de la siguiente manera:

$$log_{return} = \log(\frac{y_t}{y_{t-1}})$$

La serie temporal de acciones de Amazon ya se cargó para usted como amazon. Puede calcular el retorno de registro de este DataFrame sustituyendo:

- $y_t \longrightarrow$ Amazon
- $y_{t-1} \longrightarrow$ Amazon.shift(1)
- $\log() \longrightarrow$ np.log()

En este ejercicio, comparará la transformación log-return y la diferencia de primer orden de la serie temporal de acciones de Amazon para encontrar cuál es mejor para hacer que la serie temporal sea estacionaria.

[10]:
```
amazon = pd.read_csv("/Users/juanisaula/Downloads/amazon_close.
 ↪csv",index_col="date",parse_dates=True)

amazon_diff = amazon.diff().dropna()
amazon_diff = amazon_diff.dropna()

# Corremos el test y lo imprimimos
result_diff = adfuller(amazon_diff['close'])
print(result_diff)
```

```
(-7.2035794888112905, 2.331271725486561e-10, 23, 1234, {'1%':
-3.435660336370594, '5%': -2.863885022214541, '10%': -2.568018522153254},
10764.626718933836)
```

[11]:
```
import numpy as np

# Calculamos la primera diferencia y eliminamos los NaN con dropna
amazon_diff = amazon.diff()
amazon_diff = amazon_diff.dropna()

# Corremos el test y lo imprimimos
result_diff = adfuller(amazon_diff['close'])
print(result_diff)

# Calculams log-return y eliminamos los NaN
amazon_log = np.log(amazon)
amazon_log = amazon_log.dropna()

# Corremos el test y lo imprimimos
result_log = adfuller(amazon_log['close'])
print(result_log)
```

```
(-7.2035794888112905, 2.331271725486561e-10, 23, 1234, {'1%':
-3.435660336370594, '5%': -2.863885022214541, '10%': -2.568018522153254},
10764.626718933836)
(-0.7739778922526589, 0.8266466281503939, 0, 1258, {'1%': -3.4355588184378574,
'5%': -2.8638402312881497, '10%': -2.5679946684494275}, -6249.367882443734)
```

1.2 Introducción a los modelos AR, MA y ARMA.

Ahora que sabe cómo preparar sus datos, profundicemos directamente en los modelos. Discutiremos los modelos AR y MA y cómo estos se combinan en modelos ARMA.

1.2.1 Modelo AR

En un modelo autorregresivo, retrocedemos los valores de la serie temporal contra los valores anteriores de esta misma serie temporal. La ecuación para un modelo AR simple es:

$$y_t = a_1 y_{t-1} + \epsilon_t$$

El valor de la serie temporal en el tiempo t es a una vez el valor de la serie temporal en el paso anterior. También hay un término impactante, ϵ_t. El término choque es ruido blanco, lo que significa que cada choque es aleatorio y no está relacionado con los otros choques de la serie.

a_1 es el coeficiente autorregresivo en el desfase uno. Compare esto con una regresión lineal simple donde la variable dependiente es y_t y la variable independiente es y_{t-1}. El coeficiente a_1 es solo la pendiente de la línea y los choques (ϵ_t) son los residuos de la línea.

$$y_t = a_1 y_{t-1} + \epsilon_t$$

es un modelo AR de primer orden (AR(1)). El orden del modelo es el número de retrasos de tiempo utilizados. Un modelo AR de orden dos tiene dos coeficientes autorregresivos y tiene dos variables independientes, usamos p para referirnos al orden del modelo AR. Esto significa que tenemos p coeficientes autorregresivos y usamos p rezagos (lags).

$$y_t = a_1 y_{t-1} + \epsilon_t \longrightarrow AR(1)$$
$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \epsilon_t \longrightarrow AR(2)$$
$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + ... + a_p y_{t-p} + \epsilon_t \longrightarrow AR(p)$$

1.2.2 Modelo MA

En un modelo de promedio móvil, hacemos una regresión de los valores de la serie de tiempo contra los valores de choque anteriores de esta misma serie de tiempo. La ecuación para un modelo MA simple se muestra a continuación,

$$y_t = m_1 \epsilon_{t-1} + \epsilon_t$$

El valor de la serie temporal es m veces el valor del choque en el paso anterior; más un término de choque para el paso de tiempo actual. Este es un modelo MA de primer orden. Nuevamente, el orden del modelo significa cuántos retrasos de tiempo usamos. Un modelo MA dos incluiría choques de hace uno y dos pasos. Más generalmente, usamos q para indicar el orden del modelo MA.

$$y_t = m_1 \epsilon_{t-1} + \epsilon_t \longrightarrow MA(1)$$
$$y_t = m_1 \epsilon_{t-1} + m_2 \epsilon_{t-2} + \epsilon_t \longrightarrow MA(2)$$
$$y_t = m_1 \epsilon_{t-1} + m_2 \epsilon_{t-2} + ... + m_q \epsilon_{t-q} + \epsilon_t \longrightarrow MA(q)$$

1.2.3 Modelo ARMA

Un modelo ARMA es una combinación de los modelos AR y MA. Se realiza una regresión de la serie temporal sobre los valores anteriores y los términos de choque anteriores. ARMA = AR + MA. Este es un modelo ARMA(1,1)

$$y_t = a_1 y_{t-1} + m_1 \epsilon_{t-1} + \epsilon_t$$

De manera más general, usamos ARMA(p,q) para definir un modelo ARMA. La p nos dice el orden de la parte autorregresiva del modelo y la q nos dice el orden de la parte de la media móvil.

Usando el paquete statsmodels, podemos ajustar modelos ARMA y crear datos ARMA. Tomemos este modelo ARMA(1,1)

$$y_t = 0.5 y_{t-1} + 0.2 \epsilon_{t-1} + \epsilon_t$$

. Digamos que queremos simular datos con estos coeficientes.

[12]:
```
from statsmodels.tsa.arima_process import arma_generate_sample

ar_coefs = [1, -0.5]
ma_coefs = [1,  0.2]

y = arma_generate_sample(ar_coefs, ma_coefs, nsample = 100, scale = 0.5)
```

1.2.4 Ajuste y Modelo ARMA

El ajuste se cubre en el próximo capítulo, pero aquí hay un vistazo rápido a cómo podríamos ajustar estos datos. Primero importamos la clase del modelo ARMA. Instanciamos el modelo, lo alimentamos con los datos y definimos el orden del modelo. Entonces finalmente ajustamos:

from statsmodels.tsa.arima_model import ARMA Creamos el modelo model = ARMA(y, order=(1,1)) Ajustamos el modelo results = model.fit()

Pregunta Si tenemos, ar_coefs = [1, 0.4, -0.1] y ma_coefs = [1, 0.2] ¿Cuál sería el orden de los datos?

Respuesta: ARMA(2,1)

¿cuál es el coeficiente AR lag-1?

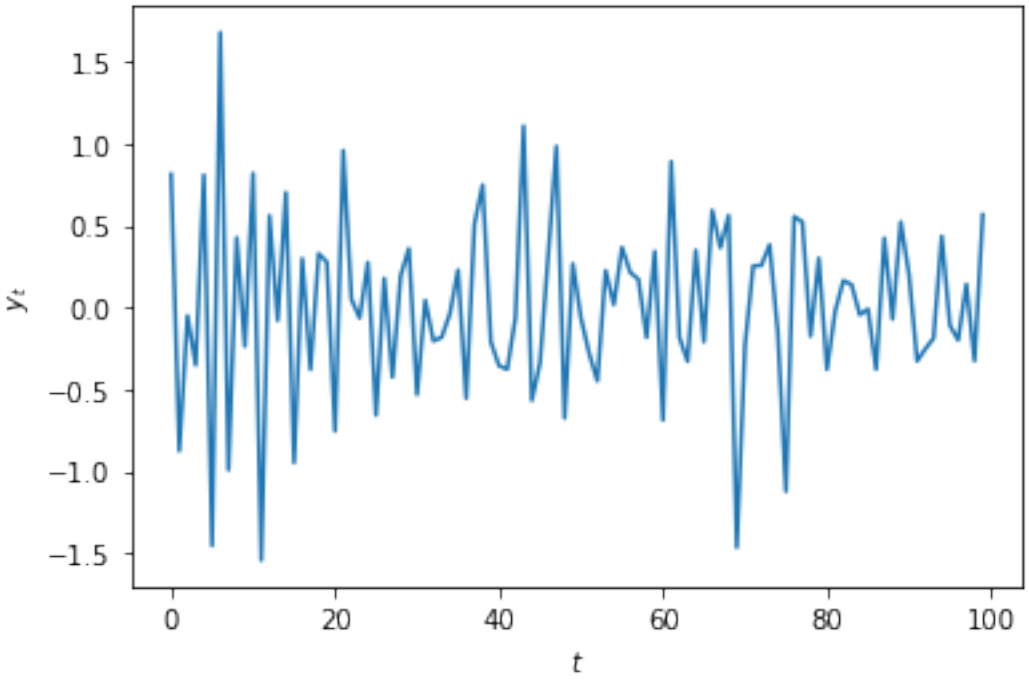
Respuesta: -0.4

¿Cuál de estos modelos es equivalente a un modelo AR(1)?

Respuesta: ARMA(1,0)

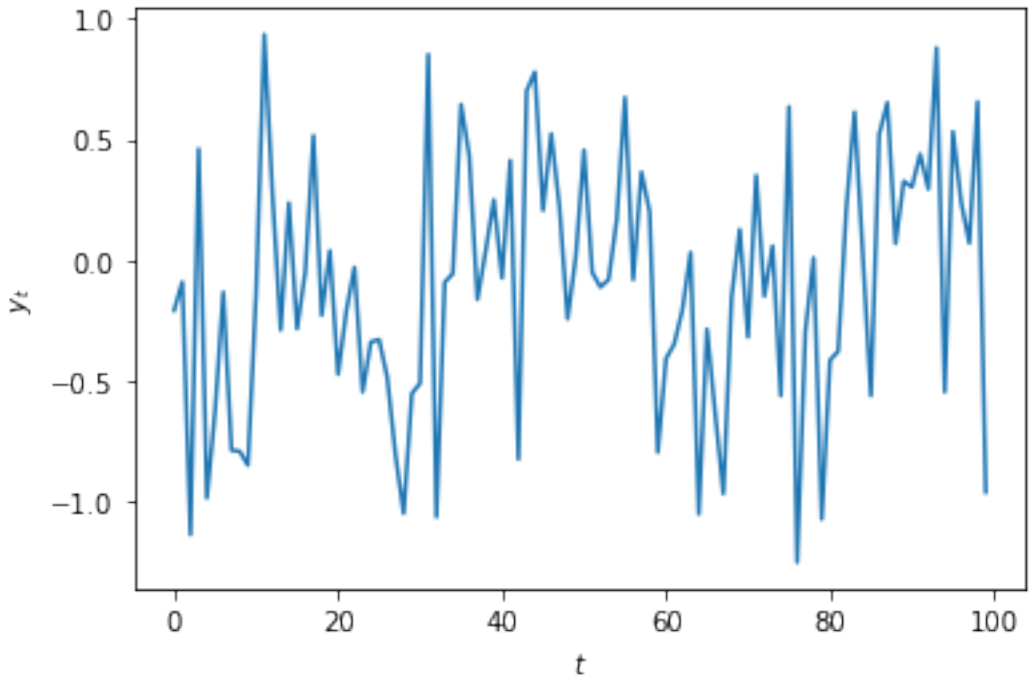
[13]:
```
"""
GENERACION DE DATOS ARMA
"""

# Importe la función de generación de datos y establezca semillas aleatorias
from statsmodels.tsa.arima_process import arma_generate_sample
np.random.seed(1)

# Establecer coeficientes
ar_coefs = [1]
ma_coefs = [1, -0.7]
```

```
# Generacion de datos
y = arma_generate_sample(ar_coefs, ma_coefs, nsample=100, scale=0.5)

plt.plot(y)
plt.ylabel(r'$y_t$')
plt.xlabel(r'$t$')
plt.show()
```

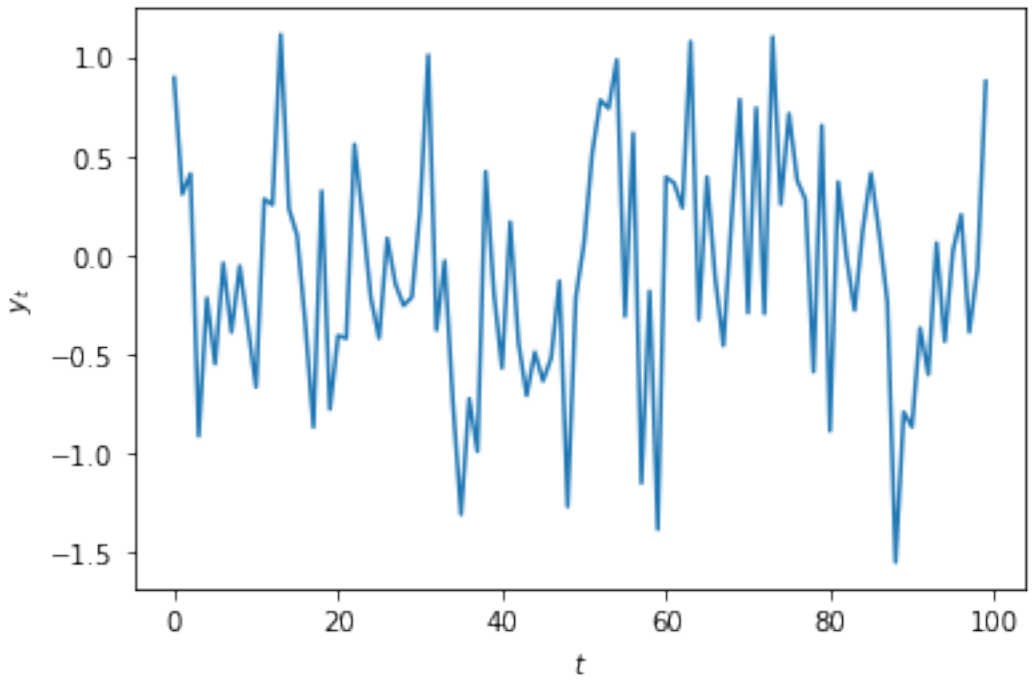


[14]:
```
"""
Establezca los coeficientes para un modelo AR(2)
con coeficientes AR lag-1 y lag-2 de 0,3 y 0,2
respectivamente.
"""
from statsmodels.tsa.arima_process import arma_generate_sample
np.random.seed(2)

ar_coefs = [1, -0.3, -0.2]
ma_coefs = [1]

y = arma_generate_sample(ar_coefs, ma_coefs, nsample=100, scale=0.5)

plt.plot(y)
plt.ylabel(r'$y_t$')
plt.xlabel(r'$t$')
```

```
plt.show()
```



[15]:
```
"""
Establecer los coeficientes para un modelo dado
y_{t} = -0.2y_{t-1} + 0.3e_{t-1} + 0.4e_{t-2} + e_{t}
"""
from statsmodels.tsa.arima_process import arma_generate_sample
np.random.seed(3)

ar_coefs = [1,0.2]
ma_coefs = [1,0.3,0.4]

y = arma_generate_sample(ar_coefs, ma_coefs, nsample=100, scale=0.5)

plt.plot(y)
plt.ylabel(r'$y_t$')
plt.xlabel(r'$t$')
plt.show()
```



¡Genial, entiendes el orden del modelo! Comprender el orden es importante cuando se trata de ajustar modelos. Siempre deberá seleccionar el orden del modelo que ajusta a sus datos, sin importar cuáles sean esos datos.