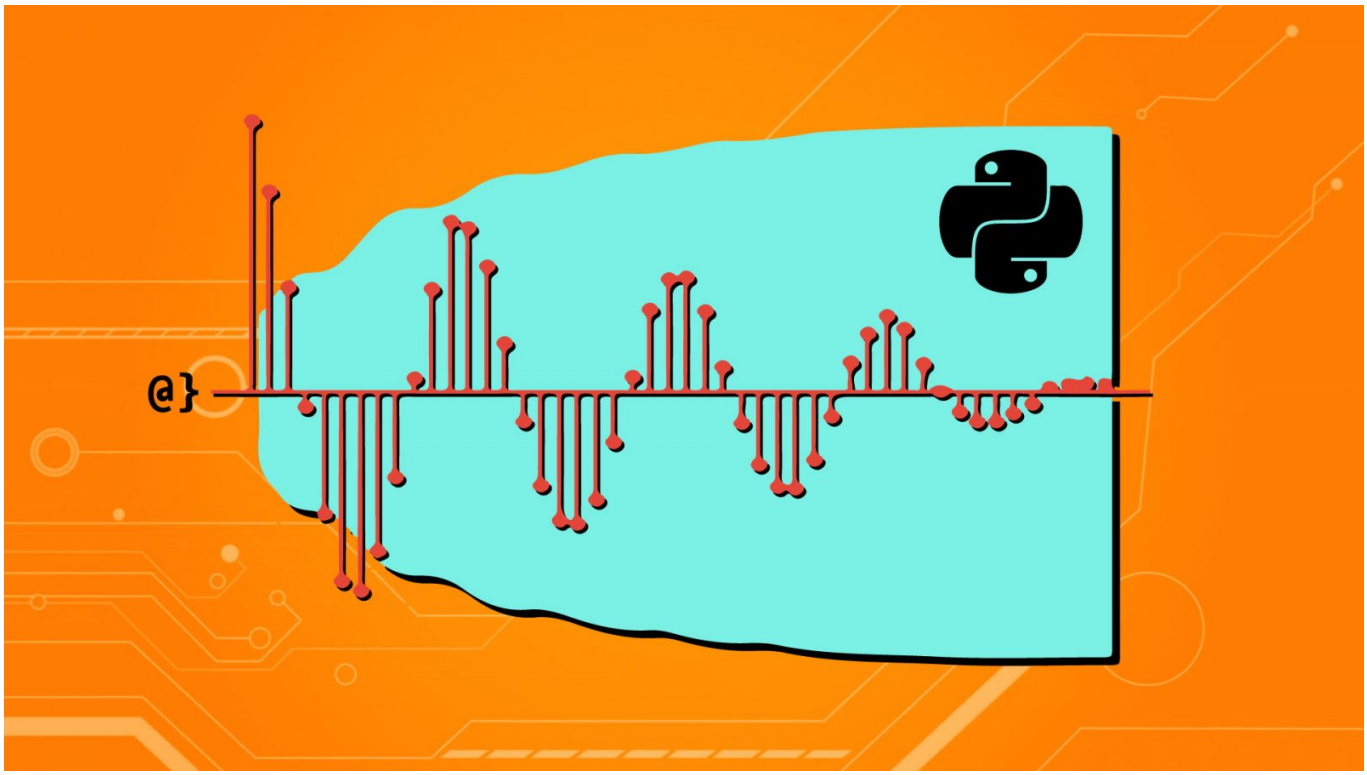
Asignación Reproduccir la siguiente pagina [ACF - Python]

Autocorrelación de datos de series temporales en Python



La autocorrelación (ACF) es un valor calculado que se usa para representar qué tan similar es un valor dentro de una serie de tiempo a un valor anterior. La biblioteca Statsmoldels simplifica mucho el cálculo de la autocorrelación en Python. Con unas pocas líneas de código, se pueden obtener conocimientos prácticos sobre los valores observados en los datos de series temporales. El ACF se puede utilizar para identificar tendencias en los datos y la influencia de los valores observados anteriormente en una observación actual. Las aplicaciones del ACF son amplias, pero sobre todo se pueden utilizar para el procesamiento de señales, la previsión meteorológica y el análisis de valores. ¡A veces, incluso puedes descubrir tendencias ocultas que son cualquier cosa menos intuitivas!

TL; DR : encontrar la autocorrelación en Python para datos de series temporales es fácil cuando se usa el `statsmodels plot_acf` funcionar como tal:

Tabla de contenido

```
 1 import pandas as pd
 2 from matplotlib import pyplot as plt
 3 from statsmodels.graphics.tsaplots import plot_acf
 4 # Have some time series data (via pandas)
 5 data = pd.read_csv('time-series.csv')
 6 # Select relevant data, index by Date
 7 >>> data = data[['Date', 'Observations']].set_index(['Date'])
 8 # Calculate the ACF (via statsmodel)
 9 >>> plot_acf(data)
10 # Show the data as a plot (via matplotlib)
11 plt.show()
```

1. Introducción

La autocorrelación, también llamada correlación en serie , es utilizada por comerciantes de acciones, meteorólogos, químicos y más para pronosticar valores futuros dados datos históricos de series temporales. Ese es un aspecto crucial para calcular tanto la autocorrelación como las autocorrelaciones parciales: datos anteriores. Este tipo de análisis regresivo se utiliza para ayudar a predecir precios futuros dentro de un intervalo de confianza (típicamente 95%) y relaciona un valor actual con los anteriores.

La autocorrelación estima la influencia de todos los valores observados en el pasado sobre el valor observado actualmente. Esto difiere de la autocorrelación parcial en la que solo se mide un único valor observado en el pasado para determinar la influencia en el valor observado actualmente.

2. Cálculo de la autocorrelacíon en Python

Aprender a encontrar la autocorrelación en Python es bastante simple, pero con algunas consideraciones adicionales, veremos cómo y dónde se puede aplicar esta función y dónde y cuándo puede fallar. Primero, repasemos algunas definiciones rápidas:

01: Retraso : el número de observaciones anteriores medidas durante la autocorrelación.

02: Correlación positiva : una relación en la que el aumento de un valor predice un aumento en otro.

03: Correlación negativa : una relación en la que el aumento de un valor predice una disminución en otro.

04: Intervalo de confianza : un rango calculado de valores en el que probablemente contendría un valor desconocido para los datos muestreados.

05: Nivel de confianza : probabilidad de que un intervalo de confianza contenga un valor observado.

Comprender estos términos no es esencial para crear un gráfico de autocorrelación en Python, ¡pero mejorará en gran medida nuestra capacidad para interpretar ese gráfico! En este artículo, usaremos el `statsmodel` función plot_acf de la biblioteca para analizar datos de series temporales y trazar la función de autocorrelación. estaremos usando `Pandas` , `Statsmodel` , `matplotlib` , por supuesto, ¡Python!

Nota : Para una discusión más profunda, más matemática, sobre cómo calcular los valores de autocorrelación, sugiero leer este artículo . Nuestra discusión aquí no tocará los puntos más finos del cálculo de la función ACF y más sobre la representación visual y la interpretación

3. Paso 1: Obtener datos de series temporales

El primer paso es obvio: necesitamos obtener algunos datos. La generación de datos de series temporales aleatorias puede ser una herramienta útil para explorar herramientas de análisis como `statsmodels` y `matplotlib` . Sin embargo, los datos generados aleatoriamente no reflejarán las tendencias que aparecerán en el análisis autorregresivo. Teniendo en cuenta este conocimiento, utilizaremos datos históricos de acciones de $TSLA para este artículo. Echemos un vistazo a nuestros datos:

```
 1 import pandas as pd
 2 # Read in CSV data to Dataframe
 3 df = pd.read_csv('TSLA.csv')
 4 # Drop all columns but 'Date' and 'Adj Close', reindex using 'Date'
 5 df = df[['Date', 'Adj Close']].set_index(['Date'])
 6 # View data
 7 >>> df
 8              Adj Close
 9 Date
10 2020-01-02   86.052002
11 2020-01-03   88.601997
12 2020-01-06   90.307999
13 2020-01-07   93.811996
14 2020-01-08   98.428001
15 ...                ...
16 2021-06-23  656.570007
17 2021-06-24  679.820007
```

```
18 2021-06-25  671.869995
19 2021-06-28  688.719971
20 2021-06-29  680.760010
21 [376 rows x 1 columns]
```

Nota : Los datos de precios históricos provienen de finance.yahoo.com , pero también se pueden descargar desde Github aquí .

Como podemos ver, ahora tenemos 376 precios registrados desde el 01-01-2020 hasta el 30-06-2021, lo que refleja un aumento porcentual del 691,103%. No está mal: si tan solo hubiéramos estado haciendo algo de autocorrelación a principios de 2020, ¡podríamos haber estado surfeando esa ola!

Paso 2: Inspeccionar y limpiar datos

Antes de realizar una autocorrelación en nuestra serie temporal, debemos inspeccionar los datos en busca de valores faltantes. Afortunadamente, Pandas ofrece varias herramientas convenientes para encontrar y reemplazar valores faltantes en datos de series temporales. No entraremos en todas las opciones y consideraciones para tal operación en este artículo. En cambio, hagamos una verificación rápida para ver si faltan valores:

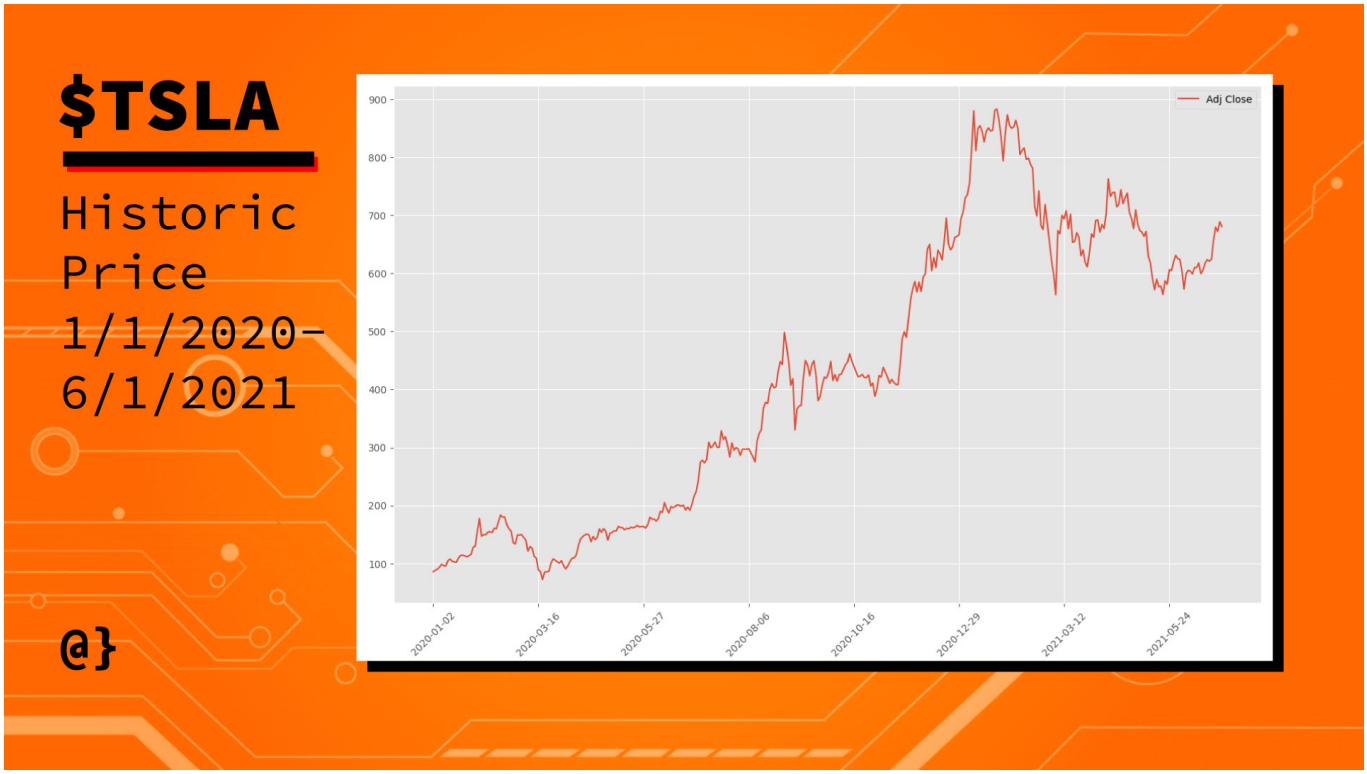
```
1 # Comprobar si hay valores nulos o NaN en los datos
2 >>> df. es nulo () . suma ()
3 # Resultado
4 Ajustar cerrar     0
5 tipo: int64
```

Al usar Panda `isnull().sum()` método se nos dice que nuestra única columna sin índice Adj Close contiene cero `null` o `NaN` valores. Esto significa que podemos pasar al siguiente paso y comenzar a visualizar nuestros datos.

Nota : consulte este artículo para obtener una discusión detallada sobre otras formas de encontrar y reemplazar datos faltantes usando Pandas.

Paso 3: Visualizar datos

Antes de calcular una autocorrelación, será útil comprender cómo se ven visualmente nuestros datos. Esto puede ayudar a detectar posibles tendencias, informarnos si es probable que los datos no tengan una correlación lineal o hacer evidente una serie de otros problemas que podrían afectar la forma en que elegimos avanzar. En algunos casos, visualizar los datos puede ayudarnos a darnos cuenta de que debemos dar un paso atrás y reevaluar nuestras prácticas de inspección y limpiar mejor los datos. Para este paso, usaremos un diagrama de línea simple a través `matplotlib` :



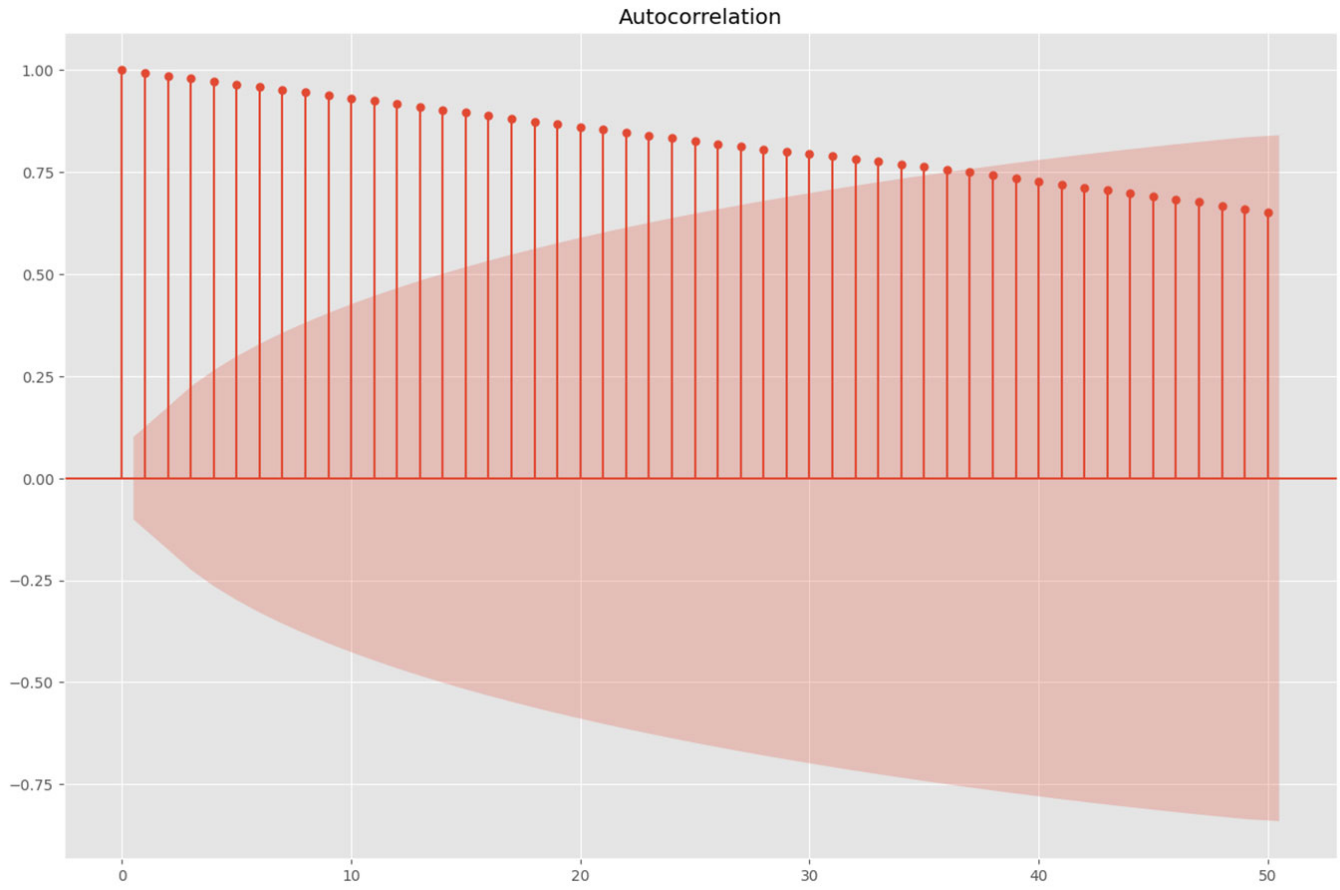
Esto nos permite saber que nuestros datos reflejan una tendencia ascendente constante que habíamos adivinado después de calcular el aumento porcentual de nuestros valores anteriormente. Dado que no hay caídas masivas en los valores graficados, diría que nuestra verificación de datos faltantes fue exitosa. ¡Ahora podemos pasar a la autocorrelación!

Paso 4: Realizar Autocorrelación y Visualización

Ahora que tenemos confianza en nuestros datos, podemos proceder a generar una visualización de autocorrelación usando `statsmodel`, `pandas`, `matplotlib`, y `Python` Estos módulos se han abstraído en gran medida y todo este proceso se puede realizar en solo unas pocas líneas de código:

```
from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt
# Use the Autocorrelation function
# from the statsmodel library passing
# our DataFrame object in as the data
# Note: Limiting Lags to 50
plot_acf(data=data, lags=50)
# Show the AR as a plot
plt.show()
```

Vemos que se necesitan dos importaciones: `pyploty` y `statsmodels` `plot_acf` unción para crear la visualización. Afortunadamente, estas dos bibliotecas funcionan bien juntas y el `pyplot` el gráfico está implícitamente referenciado. Esto da como resultado la siguiente trama:



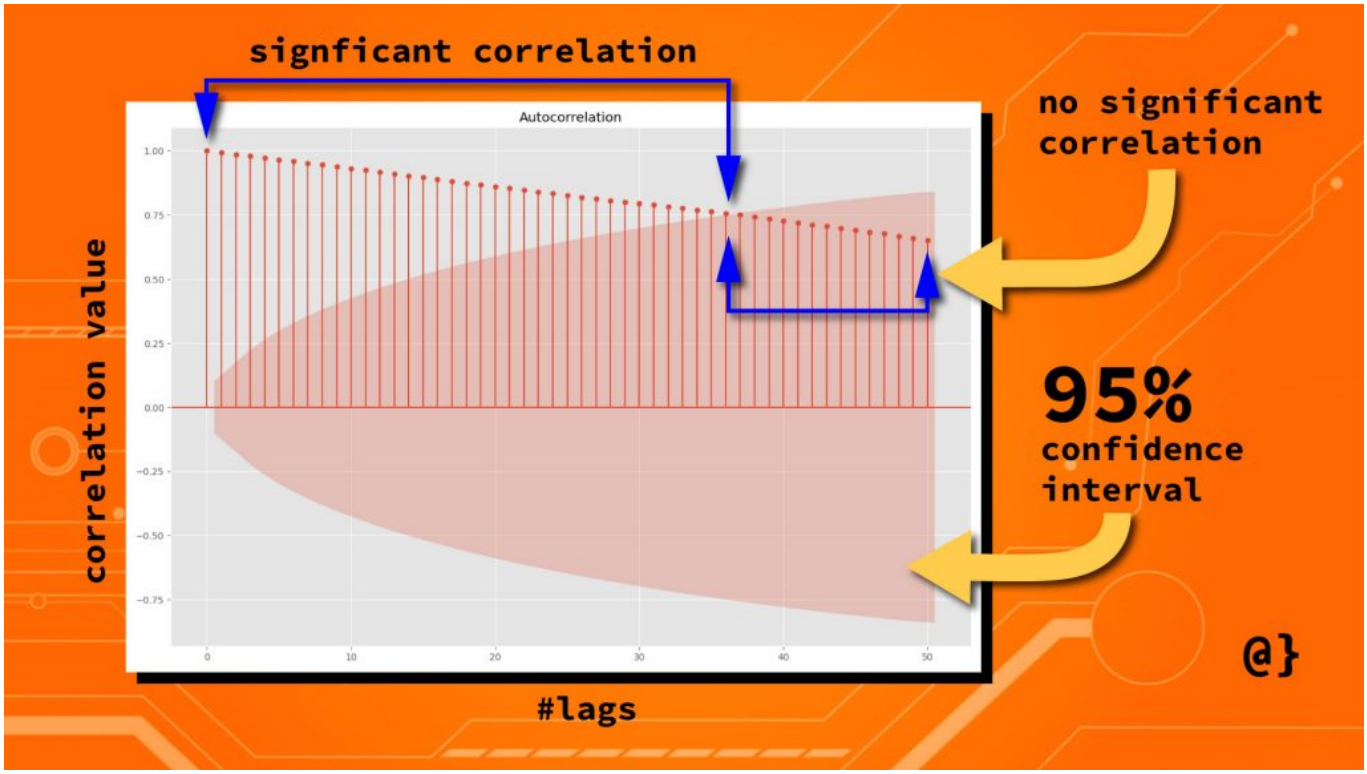
¡Esto parece bastante interesante pero en última instancia no ofrece mucha utilidad sin saber cómo leerlo! La región sombreada en rojo es el intervalo de confianza con un valor predeterminado de α = 0,05. Cualquier cosa dentro de este rango representa un valor que no tiene una correlación significativa con el valor más reciente del precio.

Las líneas verticales con marcadores en su parte superior son los "retrasos" que representan un número específico (50, en este caso) de valores anteriores. Estos representan el valor de correlación (que se muestra en el eje y) y disminuyen a un ritmo constante a medida que aumenta su proximidad al precio actual. Ese no es el caso con todos los datos, pero sí con los nuestros.

Paso 5: Interpretación de la Autocorrelación

Esto nos permite saber que los precios anteriores influyen en el precio actual, pero la importancia de esa influencia disminuye constantemente con el tiempo. Específicamente, los valores más allá del día 36 de negociación anterior no tienen un poder predictivo significativo sobre el precio actual.

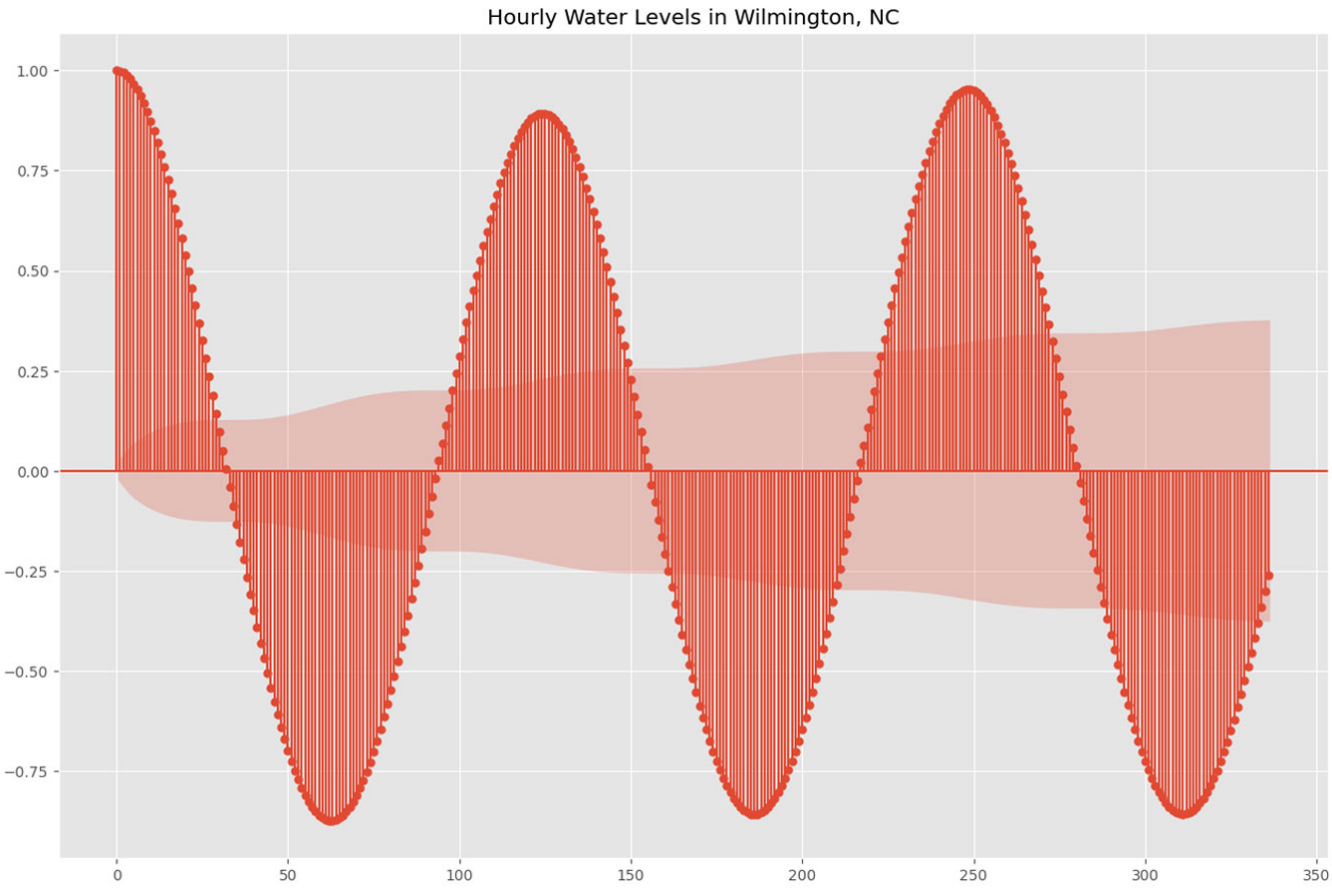
La fuerza de esta relación se mide en una escala de -1 a 1, siendo -1 una correlación negativa del 100 % y siendo 1 una correlación positiva del 100 %. Esta medida se muestra en el eje y. Considere el siguiente diagrama para una interpretación más visual:



Aplicaciones

Hemos visto con qué facilidad se pueden visualizar los datos de autocorrelación usando la biblioteca. Hemos visto cómo interpretar estas representaciones visuales para [con suerte] obtener información sobre patrones más profundos reflejados en ciertos datos observables. Nuestros datos de precios históricos para $TSLA no fueron muy emocionantes, aunque ilustraron que los precios de cierre anteriores están relacionados con los precios de cierre actuales. Consideremos algunos datos más emocionantes y reveladores

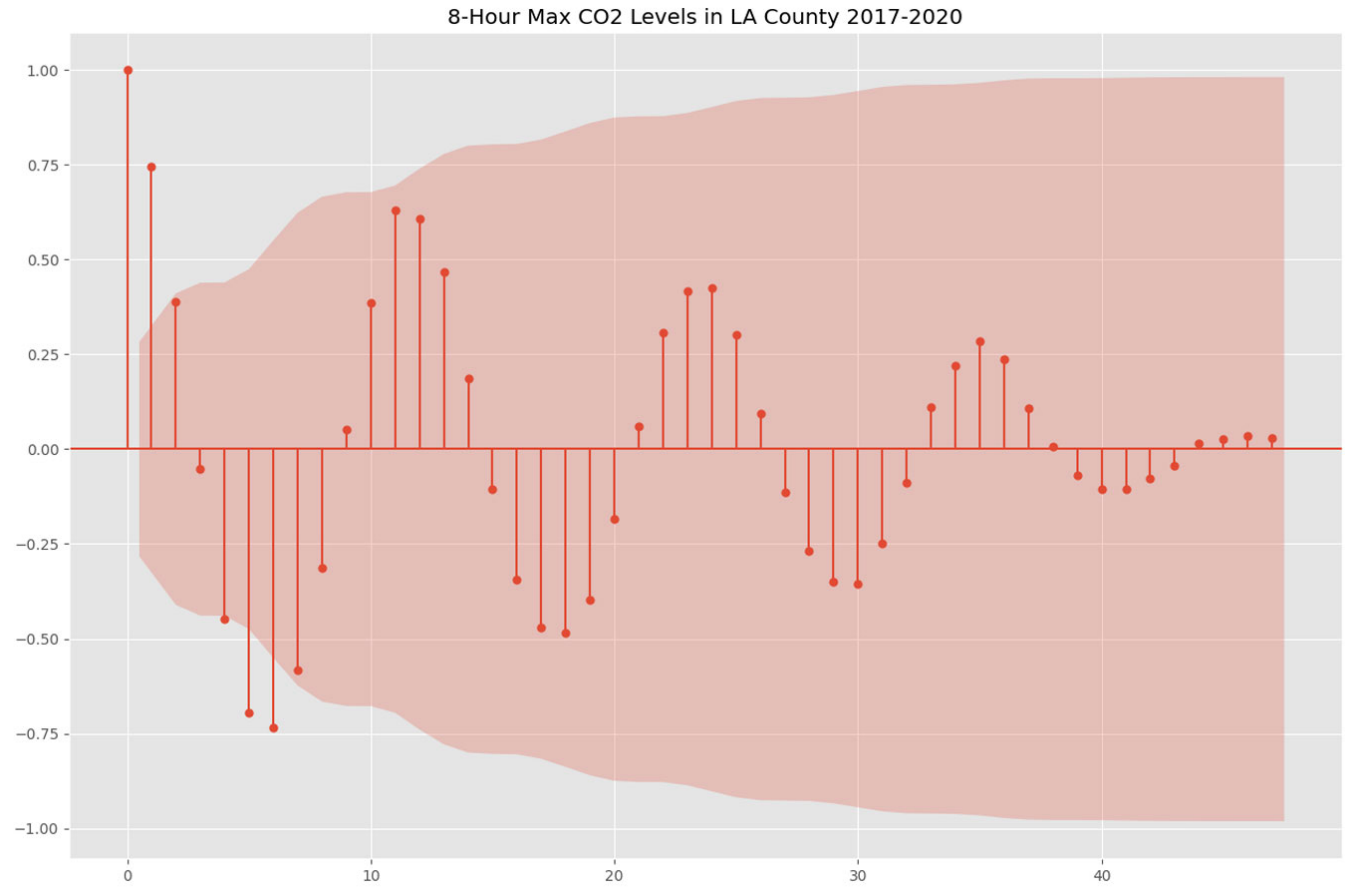
Datos de mareas Los datos meteorológicos son una aplicación común para el análisis de autocorrelación. Las influencias estacionales, regionales e incluso diarias pueden revelarse dramáticamente mediante representaciones visuales del análisis de autocorrelación. Aquí hay un vistazo a los datos de mareas medidos cada seis minutos:



Cada punto de datos representa una medida del nivel del agua registrada en intervalos de 6 minutos (240 por día). Aquí vemos surgir una fuerte tendencia de correlación positiva y negativa. Esto es representativo de lo que esperaríamos dado lo que se ha observado durante décadas de estudio de las mareas.

La contaminación del aire

La contaminación del aire es otra aplicación común para la autocorrelación. Si bien los datos de mareas representan una periodicidad muy conocida, estos datos a menudo reflejan patrones estacionales con menos datos para predecir la frecuencia o la fuerza de la correlación. El cuadro a continuación describe los niveles máximos de CO2 observados en 8 horas para el condado de Los Ángeles, California, de 2017 a 2020.



Vemos aquí que, si bien no tiene importancia estadística, existe un fuerte patrón observable en el que los valores pasados pueden usarse para pronosticar valores futuros. Estos datos han sido remuestreados para reflejar los promedios mensuales. En este caso, los valores de retraso de 6 a 7 meses reflejan una fuerte correlación negativa y los valores de retraso de 6 a 7 meses reflejan una fuerte correlación positiva.

Limitaciones

La autocorrelación es una herramienta útil para detectar patrones de periodicidad, estacionalidad u otras fuentes de influencia menos intuitivas. Ninguna técnica de pronóstico es perfecta y la autocorrelación no es una excepción. Algunas áreas bien conocidas en las que la autocorrelación puede fallar se enumeran a continuación:

Demasiada información retenida En el procesamiento de señales, los datos a menudo representan cambios sutiles o artefactos generados durante la observación. Aplicaciones como el reconocimiento de patrones de voz, el análisis de radiofrecuencia y el análisis de rayos X generan cantidades masivas de ruido. Estos puntos de datos pueden sesgar las métricas de correlación, de modo que un modelo de autocorrelación podría encontrar una influencia falsa o no capturar una influencia positiva (Fukushima, 1985; Willink, 2013)

Consideraciones finales

La autocorrelación es una herramienta útil para identificar relaciones estadísticamente significativas entre valores observados en datos lineales. Hemos visto cómo el ACF es útil para identificar tendencias estacionales o naturales, cómo se puede aplicar al análisis técnico de los datos de precios de acciones e incluso notamos algunas de sus deficiencias.

También hemos visto cómo el ACF se puede calcular y visualizar de manera rápida y efectiva en Python. Tener este tipo de herramienta estadística a mano puede ayudar a analizar e interpretar mejor los datos de manera que ayude a tomar decisiones mejor informadas. Por ejemplo, organizar su festival al aire libre en Los Ángeles cuando sus datos indican que los niveles de contaminación del aire serían más bajos.

Referencias

01: Willink, Tricia J. "Límites en la estimación de matrices de autocorrelación a partir de mediciones MIMO móviles". Revista Internacional de Antenas y Propagación , vol. 2013, 2013, págs. 1 a 6.

02: Fukushima, Teiichiro, et al. "Limitaciones de la autocorrelación en la monitorización de la frecuencia cardíaca fetal". Revista americana de obstetricia y ginecología , vol. 153, núm. 6, 1985, págs. 685–92.

Codigo funcional de la pagina ACF - Python con datos de precipitacion del 2021 en Mosquera, Cundinamarca

```
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
# Have some time series data (via pandas)
data = pd.read_csv('time-series.csv')
# Select relevant data, index by Date
>>> data = data[['Date', 'Observations']].set_index(['Date'])
# Calculate the ACF (via statsmodel)
>>> plot_acf(data)
# Show the data as a plot (via matplotlib)
plt.show()


import pandas as pd
# Read in CSV data to Dataframe
df = pd.read_csv('TSLA.csv')
# Drop all columns but 'Date' and 'Adj Close', reindex using 'Date'
df = df[['Date', 'Adj Close']].set_index(['Date'])
# View data
>>> df
```

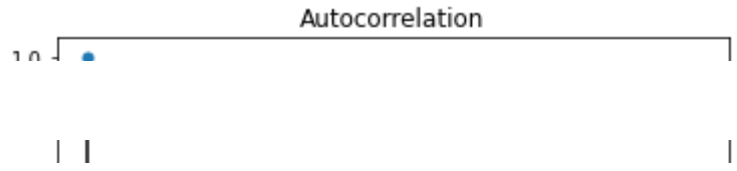
```
             Adj Close
Date
2020-01-02   86.052002
2020-01-03   88.601997
2020-01-06   90.307999
2020-01-07   93.811996
2020-01-08   98.428001
...                ...
2021-06-23  656.570007
2021-06-24  679.820007
2021-06-25  671.869995
2021-06-28  688.719971
2021-06-29  680.760010
[376 rows x 1 columns]

# Check if any null or NaN values in data
>>> df.isnull().sum()
# Result
Adj Close    0
dtype: int64

from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt
# Use the Autocorrelation function
# from the statsmodel library passing
# our DataFrame object in as the data
# Note: Limiting Lags to 50
plot_acf(data=data, lags=50)
# Show the AR as a plot
plt.show()
```

```
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
df=pd.read_excel ('/content/drive/MyDrive/Datos marengo 2021/Precipitacion diaria mosqu
df = df[['Fecha', 'Valor']].set_index(['Fecha'])
df
df.isnull().sum()
plot_acf(df)
plt.show
##En este caso se puede observar que los datos de precipitacion diaria no estan autocor
```

```
<function matplotlib.pyplot.show>
```



Convertir ºC a Kelvin

```
1 CaK=lambda numero: numero+273.15
2 CaK(20)
```

```
293.15
```

Calcular el CV

```
1 import numpy as np
2 CV= lambda x: np.std(x,ddof=1)/np.mean(x)*100
3 data = [12, 16, 13, 16, 17, 21, 11, 15, 14, 18, 22, 9, 8, 7, 19, 14]
4 CV(data)
```

```
30.428307324642383
```

Seleccionar un tamaño de muestra

```
1
2 ################## matriz de datos
3 import pandas as pd
4 import numpy as np
5 def prog(r, n, a1):
6   an = a1 + r*(n-1)
7   seq = np.arange(start=a1, stop=an, step=r)
8   return seq
9
10 prog(r=7, n=20, a1=15)
11 np.random.seed(472)
12
13 df = pd.DataFrame({
14     'x': np.sort(np.random.normal(loc = 4, scale = 1, size=96)),
15     'y': np.sort(np.random.normal(loc=4.5, scale=1.2, size=96)),
16 })
17 ########################
18
19 Muestra=lambda x:df.sample(n=x)
20 Muestra(12)
```

	x	y
72	4.574063	5.124028
85	5.290240	5.918715
88	5.668290	6.275213
7	2.807086	2.439520
35	3.657533	3.873315
16	3.189569	3.271792
69	4.484093	5.088295
54	4.046274	4.500201
79	5.008548	5.469825
13	3.080553	2.931073
47	3.934366	4.293883

Convertir absorbancia en transmitancia (%)

```
1 TaA=lambda A:10**(2-A)
2 TaA(0.5)
```

```
31.622776601683793
```

Convertir coordenadas rectangulares a polares

```
1 rpolar=lambda x,y:np.sqrt(x**2+y**2)
2 rpolar(1,4)
3
4
```

```
4.123105625617661
```

```
1 tpolar=lambda x,y:np.arctan2(y,x)
2 tpolar(1,4)
```

```
1.3258176636680326
```

✓ 0 s completado a las 21:12