Logaritmos

Esta es una función muy práctica para el cálculo de los retornos logarítmicos, dado que la eficiencia de numpy para el cálculo es imbatible yo la he usado mucho

Aquí un ejemplo

```
precios = np.array([116.72, 115.47, 116.17, 116.92, 118.23, 117.51, 116.95, 118.59,
                    115.09, 117.11, 117.77, 117.42, 117.92, 115.87, 115.7 , 119.72])

rend_log = np.log(precios[1:]/precios[:-1])
rend_log
```

```
array([-0.01076715,  0.00604388,  0.0064353 ,  0.01114194, -0.00610844,
       -0.00477694,  0.01392567, -0.02995773,  0.01739923,  0.00561991,
       -0.00297632,  0.00424918, -0.01753756, -0.00146824,  0.03415505])
```

Lo del precios[1:] es desde el segundo al ultimo, y precios[:-1] es desde el primero al anteúltimo, por lo tanto el cociente siempre hace: precios $_{[i+1]}$/ precios $_{[i]}$
Y la serie tendrá un valor menos que precios

Retornos Logarítmicos

Voy a hacer un mini paréntesis a numpy, o no tanto, para repasar el tema de los retornos logarítmicos y su uso correcto, porque me lo han preguntado mucho en las clases y veo que muchos inversores tienen alguna idea del tema pero les falta comprender bien esto y como no es un tema menor prefiero perder un par de páginas del libro pero que quede la base sólida de cuando, como y por que usar o no usar retornos lineales y logarítmicos

Primero veamos el cálculo de ambos retornos:

```
precios = np.array([116.72, 115.47, 116.17, 116.92, 118.23, 117.51, 116.95, 118.59,
                    115.09, 117.11, 117.77, 117.42, 117.92, 115.87, 115.7 , 119.72])

ret =  precios[1:]/precios[:-1] -1
ret_log = np.log(precios[1:]/precios[:-1])

ret, ret_log
```

```
(array([-0.01070939,  0.00606218,  0.00645606,  0.01120424, -0.00608982,
        -0.00476555,  0.01402309, -0.02951345,  0.01755148,  0.00563573,
        -0.00297189,  0.00425822, -0.01738467, -0.00146716,  0.03474503]),
 array([-0.01076715,  0.00604388,  0.0064353 ,  0.01114194, -0.00610844,
        -0.00477694,  0.01392567, -0.02995773,  0.01739923,  0.00561991,
        -0.00297632,  0.00424918, -0.01753756, -0.00146824,  0.03415505]))
```

Como verán son muy parecidos, pero levemente difieren, ¿ahora cual es correcto? Ambos, el tema es como se usa cada uno, por ejemplo si calculara el promedio de los retornos, ambos son incorrectos, aunque el promedio de los retornos logarítmicos será mas parecido al retorno medio real

Para empezar debemos repasar una propiedad importante de los logaritmos, mil disculpas por las fórmulas en realidad es un repaso del colegio secundario, me propuse no usar formalismos ni formulerío e ir a lo práctico pero un mínimo de fórmulas es necesario

$$\log_e(xy) = \log_e(x) + \log_e(y)$$

La propiedad se cumple en cualquier base pero vamos a utilizar la base “e” o dicho de otra forma los logaritmos naturales que se escriben como “ln” aunque en numpy cuando hablemos de np.log ya de por si se refiere a los logaritmos naturales, si quisiera los decimales debería llamar en numpy al método np.log10(), o np.log2() para los de base 2.

Bueno, imaginemos que “x” es el cociente entre el precio de un día y el día anterior, lo mismo “y”
La fórmula anterior quedaría algo así:

$$\log_e(\frac{P_{i+2}}{\cancel{P_{i+1}}}\frac{\cancel{P_{i+1}}}{P_i}) = \log_e(\frac{P_{i+2}}{P_{i+1}}) + \log_e(\frac{P_{i+1}}{P_i})$$

Y si tuviéramos 3 elementos:

$$\log_e(\frac{P_{i+3}}{\cancel{P_{i+2}}}\frac{\cancel{P_{i+2}}}{\cancel{P_{i+1}}}\frac{\cancel{P_{i+1}}}{P_i}) = \log_e(\frac{P_{i+3}}{P_{i+2}})\log_e(\frac{P_{i+2}}{P_{i+1}}) + \log_e(\frac{P_{i+1}}{P_i})$$

Generalizando:

$$\log_e \frac{P_{i+n}}{P_i} = \sum_{j=0}^{n} \log_e \frac{P_{i+j+1}}{P_{i+j}}$$

O sea que la sumatoria de todos los retornos logarítmicos de los precios P es igual a el retorno logarítmico del último precio P de la serie respecto al primero

Esta propiedad es muy práctica porque si divido a ambos términos por “n” obtengo:

$$\frac{1}{n}.\log_e \frac{P_{i+n}}{P_i} = \frac{1}{n}.\sum_{j=0}^{n}\left(\log_e \frac{P_{i+j+1}}{P_{i+j}}\right)$$

Que aplicando otra propiedad de los logaritmos: k . log x = log x^k
Me queda:

$$\log_e\left(\frac{P_{i+n}}{P_i}\right)^{\frac{1}{n}} = \frac{1}{n}.\sum_{j=0}^{n}\left(\log_e \frac{P_{i+j+1}}{P_{i+j}}\right)$$

Ahora todo lo que está a la derecha es el promedio de los retornos logarítmicos para cada P_i
Entonces, llamemos $\bar{r}$ al promedio de los retornos logarítmicos y tenemos:

$$\log_e \left(\frac{P_{i+n}}{P_i} \right)^{\frac{1}{n}} = \bar{r}$$

O lo que es lo mismo, expresado como potencia:

$$\boxed{e^{\bar{r}} = \left(\frac{P_{i+n}}{P_i} \right)^{\frac{1}{n}}}$$

Que es a la fórmula que quería llegar, o sea que la constante “e” elevada al promedio de los retornos logarítmicos de una serie es igual al cociente entre el último y el primer valor de la serie elevado a la 1/n, siendo “n” la cantidad de valores de la serie.

O sea que si tengo el promedio de los retornos logarítmicos y el valor inicial (Vi) y la cantidad de retornos (n), puedo obtener el valor final (Vf) con la siguiente fórmula

$$\boxed{V_f = e^{n\bar{r}} . V_i}$$

Corroboremos esta última fórmula

```
import math

precios = np.array([116.72, 115.47, 116.17, 116.92, 118.23, 117.51, 116.95, 118.59,
                    115.09, 117.11, 117.77, 117.42, 117.92, 115.87, 115.7 , 119.72])

ret =  precios[1:]/precios[:-1] -1
ret_log = np.log(precios[1:]/precios[:-1])

math.e ** (len(ret_log)*ret_log.mean()) * precios[0]
```

```
119.72
```

Efectivamente me da 119.72 que era el valor final, a partir de “n” el promedio de los ret_log y el valor inicial precios[0]

Ahora es el promedio de los retornos logarítmicos equivalente el retorno medio que de aplicarlo “n” veces me daría el mismo valor final: NO! (ojo mucho menos lo es el promedio de los retornos lineales)

¿Y cual es entonces el retorno promedio que aplicado “n” veces al valor inicial me de el final?
Calculémoslo, partamos de la última fórmula y restemos 1 a ambos términos

$$\frac{V_f}{V_i} - 1 = e^{n\bar{r}} - 1$$

Obvio que “n” por el promedio de la serie de ret_log, es lo mismo que la suma de la serie ret_log
Así que chequiemos esto también:

```
import math

precios = np.array([116.72, 115.47, 116.17, 116.92, 118.23, 117.51, 116.95, 118.59,
                    115.09, 117.11, 117.77, 117.42, 117.92, 115.87, 115.7 , 119.72])

ret =  precios[1:]/precios[:-1] -1
ret_log = np.log(precios[1:]/precios[:-1])

precios[-1]/precios[0]-1, math.e**ret_log.sum()-1
```

```
(0.025702535983550323, 0.025702535983550323)
```

Lo que hice fue calcular ambos términos de la igualdad para corroborar que la fórmula es correcta

Ahora ¿Qué significa ese 0.025702..?
¿Es el retorno medio que aplicado “n” veces a Vi me da Vf?
-No

¿Y que es entonces?
- Es el retorno final equivalente a el acumulado compuesto de todos los retornos

Entonces ¿Cuál es el retorno medio tal que aplicado “n” veces a Vi me de por resultado Vf?

Bueno, ya casi lo tenemos, partiendo de la fórmula anterior, lo resumo así no se hace embolante:

Será r_c la tasa de rendimiento compuesto media que buscamos:

$$\frac{V_f}{V_i} = (r_c + 1)^n = e^{n\bar{r}} \quad \Rightarrow \quad r_c = (e^{n\bar{r}})^{\frac{1}{n}} - 1 \quad \Rightarrow \quad \boxed{r_c = \sqrt[n]{e^{n\bar{r}}} - 1}$$

Como siempre, corroboramos:

```
r_c = (math.e ** ret_log.sum())**(1/len(ret_log))-1
precios[0]*(1+r_c)**(len(ret_log))
```

```
119.71999999999997
```

Uso ret_log.sum() que es lo mismo que el promedio de los ret_log * n

Entonces resumiendo, tenemos:

- retorno lineal medio
- retorno logarítmico medio
- retorno compuesto medio

Obviamente ninguno es igual a ninguno, y dependiendo como lo usemos en nuestros cálculos será correcto o no. Lo que si les digo desde ya que el retorno lineal medio no sirve casi nunca, es muy tramposo, es decir sirve para comparar una cosa con otra por ejemplo pero rara vez sirve para algo absoluto, en cambio el retorno compuesto medio es el mas representativo en todo tipo de cálculo de portafolio y posición de método de trading en la mayoría de los casos, pero insisto, depende lo que estemos calculando, no será la misma aplicación cuando rebalanceamos que cuando no rebalanceamos ni cuando tenemos un tamaño de posición siempre fija que cuando no, etc.

Bueno, acá las fórmulas en Python de todo este embrollo

```
import math, numpy as np

precios = np.array([116.72, 115.47, 116.17, 116.92, 118.23, 117.51, 116.95, 118.59,
                    115.09, 117.11, 117.77, 117.42, 117.92, 115.87, 115.7 , 119.72])


ret =  precios[1:]/precios[:-1] -1
ret_log = np.log(precios[1:]/precios[:-1])
r_c = (math.e ** ret_log.sum())**(1/len(ret_log))-1

retorno_lineal_medio = ret.mean()
retorno_log_medio = ret_log.mean()
retorno_compuesto_medio = r_c

retorno_lineal_medio,  retorno_log_medio, retorno_compuesto_medio
```

```
(0.00180227218000432, 0.0016918519179978636, 0.0016932839069112582)
```

cumsum() y cumprod()

Esta función es la misma de pandas, de hecho la mayoría de funciones de cálculo numérico de pandas o son las mismas o se apoyan en las funciones de cálculo numérico de numpy

Siguiendo con nuestro ejemplo anterior, y ahora que dominan a la perfección los conceptos de retornos lineales y logarítmicos y sus propiedades, pueden calcular como ejercicio el retorno punta a punta de una serie usando cumsum() con retornos logarítmicos o cumprod() con retornos lineales

Es un hermoso ejercicio para internalizar estos conceptos, así que les sugiero que no lean esto y lo intenten resolver, pero por si no les sale se los dejo por aca:

```
math.e ** ret_log.cumsum()[-1]-1
```

```
0.025702535983550323
```

```
((ret+1).cumprod()-1)[-1]
```

```
0.025702535983550545
```