

PYTHON PARA FINANZAS QUANT

```
def call(S0, K, r, T, sigma, q=0):
    ret = {}
    if (S0 > 0 and K > 0 and r >= 0 and T > 0 and sigma > 0):
        d1 = (math.log(S0/K) + (r - q + sigma**2/2)*T) / (sigma * math.sqrt(T))
        d2 = d1 - sigma*math.sqrt(T)
        ret['call'] = math.exp(-q*T) * S0 * normal(d1) - K*math.exp(-r*T) * normal(d2)
        ret['delta'] = math.exp(-q*T) * normal(d1)
        ret['gamma'] = normalInv(d1) / (S0 * sigma * math.sqrt(T))
        ret['vega'] = 0.01 * S0 * math.exp(-q*T) * normalInv(d1) * math.sqrt(T)
        ret['theta'] = (1/365) * (-((S0*sigma*math.exp(-q*T))/(2*math.sqrt(T))) * normalInv(d1))
        ret['rho'] = 0.01 * K * T * math.exp(-r*T) * normal(d2)
    else:
        ret['errores'] = "Se Ingresaron valores incorrectos"
    return ret

def put(S0, K, r, T, sigma, q=0):
    ret = {}
    if (S0 > 0 and K > 0 and r >= 0 and T > 0 and sigma > 0):
        d1 = (math.log(S0/K) + (r - q + sigma**2/2)*T) / (sigma * math.sqrt(T))
        d2 = d1 - sigma*math.sqrt(T)
        ret['put'] = K*math.exp(-r*T)*normal(-d1) - S0 * math.exp(-q*T) * normal(-d2)
        ret['delta'] = -math.exp(-q*T) * normal(-d1)
        ret['gamma'] = math.exp(-q*T) * normalInv(d1) / (S0 * sigma * math.sqrt(T))
        ret['vega'] = 0.01 * S0 * math.exp(-q*T) * normalInv(d1) * math.sqrt(T)
        ret['theta'] = (1/365) * (-((S0*sigma*math.exp(-q*T))/(2*math.sqrt(T))) * normalInv(d1))
        ret['rho'] = -0.01 * K * T * math.exp(-r*T) * normal(-d2)
    else:
        ret['errores'] = "Se Ingresaron valores incorrectos"
    return ret
```

Backtesting Framework

Testeando ideas trading algorítmico

t[7]



Juan Pablo Pisano
@johnGalt_is-www

Impreso en:
laimprentadigital.com.ar

índice t[7] Backtesting

Backtesting.....	13
Palabras iniciales.....	13
Que es el Backtesting.....	15
Algoritmos backtesteables y no backtesteables	15
Tipos de Bots de trading automático.....	15
A- Bots de Colocación de órdenes grandes	16
B- Bots de Colocación de órdenes para obtener mejor precio.....	17
C- Bots de Market Making	18
D- Bots de arbitrajes inmediatos.....	19
E- Bots de arbitrajes no inmediatos.....	21
F- Bots de arbitrajes estadísticos	22
G- Bots con opciones	24
H- Bots de Balanceo de carteras (optimización, risk-management)	25
I- Bots de ruteo para seguimiento de índices (fondos, etfs)	27
J- Bots de swing trading.....	27
Bots Swing-Trading del tipo Trend-Following.....	28
Problemas de los Swings Trend Following.....	29
Bots Swing-Trading del tipo Contrarians	30
Problemas de los Swings Contrarians.....	31
Bots Swing-Trading del tipo Lateralizadores	32
Problemas de los Swings Laterales.....	32
K- Bots de scalping	33
Reflexiones Retornos de stocks vs GBM.....	36
Diferencias en la práctica	36
Conclusión Tipos de bots vs Backtesting.....	39
Etapas de un backtesting.....	39
Etapa de Datos	39
Etapa de Research / PreBacktest.....	40
Etapa de Análisis y métricas.....	40
Etapa de Parametrización	40
Etapa de contrastes y validaciones	41

Etapa de datos.....	42
Fuentes de datos	42
Extracción de datos	43
Etapa de Research / PreBacktest.....	44
Data Mining	48
Look-Ahead Bias.....	48
Correlación features vs variación	49
Clasificaciones diferenciables.....	51
Construcción de indicadores	55
Indicadores sobre serie de precios.....	57
Indicadores tardíos	57
Indicadores de flujos (volumen y derivados)	58
Construcción de Osciladores acotados desde cualquier indicador	71
Retomamos Indicadores de flujos (volumen y derivados).....	73
Combinados, Construcción de indicadores propios:	77
Conteos discretos	79
Indicadores de Estacionalidad	81
Indicadores Estadísticos o paramétricos	83
Referenciales, benchmarks	85
Indicadores de sentiment.....	91
Ratios y series de Análisis fundamental	95
Exógenos, externos	98
Ejercicios de Construcción de indicadores.....	99
Respuestas a ejercicios de indicadores	100
Filtro de datos	103
Tema de dividendos.....	103
Tema de Splits	104
Spikes (aguazos)	106
Sesgos (supervivencia y selección).....	107
Introspección de Features	108
Correlación de los features con el target	110
Multicolinealidad	111
Autoregresión de Indicadores	114

Distribuciones de Features y Target.....	116
Balanceo o Desbalanceo	116
Planteo de "Racional" o Hipótesis de trabajo.....	119
El trilema de los bots de Profit.....	121
Simulación de Gatillos de Compras y Ventas	122
Armado de tabla de Acciones	125
· Armado de Tabla de Trades.....	128
Tabla de Resultados Básicos.....	132
Etapa de Análisis y Métricas.....	137
De la tabla de trades al payoff diario.....	137
Esquema discreto de tabla de trades	137
Esquema continuo basado en eventos.....	137
Enfoque Event Driven.....	138
El Paso a Paso	139
Métricas de Riesgo.....	142
Rendimientos Lineales y Logarítmicos	147
CAGR.....	147
Volatilidad	148
Sharpe Ratio	148
Sortino Ratio	151
Máximo Drawdown	151
Duración de Drawdowns.....	153
Correlación.....	154
Momentos clave en la correlación	156
Calmar Ratio	159
Asimetría y Kurtosis	160
Esperanza Matemática	161
Criterio de Kelly.....	163
Riesgo de Ruina	164
Riesgo de Ruina Asumiendo normalidad	164
Riesgo de Ruina con datos empíricos	166
Value at Risk VaR	169
VaR con valores empíricos.....	169

VaR asumiendo normalidad	170
cVaR (valor en riesgo condicional) - Expected Shortfall	170
cVaR, cálculo empírico	171
PayOff Ratio.....	173
Profit Factor	175
Rachev Ratio	176
Tail Ratio - Por quantiles	177
Tail Ratio – Por medias.....	178
Tail Ratio – Por integrales	179
Common Sense Ratio.....	179
Mas ratios de riesgo.....	181
Tablero de control, o Dashboard	182
Etapa de parametrización.....	183
1° Paso: Performance	184
2° Paso, elegir features y target y "Montecarlear"	187
Montecarlo con 2 variables a la vez	191
Simulación montecarlo múltiple de gatillos.....	196
Iterando sobre la parametrización anterior.....	200
3° Paso y más allá.....	203
Etapa de contrastes y validaciones	205
Estudio de portabilidad	205
Primeras validaciones pre-portabilidad.....	207
Segmentación previa (paréntesis)	211
Ahora sí, empezamos la portabilidad	214
Modificación de funciones	214
Primeras pruebas manuales de portabilidad	216
Inconvenientes para escalar la portabilidad	218
Análisis de Portabilidad.....	219
Primeras reflexiones	223
Entornos más realistas, más reflexiones.....	227
Comisiones	227
Derechos de mercado.....	228
Spreads, volúmenes y liquidez	229

Apartado I: Operar sin un mercado real	231
Apartado II: Front Running	232
Impuestos	232
Costos de datos/conexiones.....	233
Costos de Infraestructura	234
Suposición de Volumen operado.....	234
Otros tipos de Backtest.....	235
Asumir el tiempo discreto	236
Asumimos ejecución 100% en un precio de dado	237
Asumimos spreads simétricos en torno al precio operado	240
Sucesos con Tick by Tick	242
Traemos data Tick by Tick	244
Traemos data Libro de órdenes	245

Backtesting

Palabras iniciales

Arrancamos el octavo tomo de esta colección. Este tomo es el primero en que empezamos a "apilar" todo lo visto en tomos anteriores para darle una finalidad específica como realizar un backtest

Les soy sincero, estuve dos semanas tirando borradores de esquemas de como diagramar el esqueleto de este tomo porque es un mundo inmenso el de los backtests, en la bibliografía que hay al respecto (que luego les dejo algunas referencias) cada autor tiene su forma muy personal de encarar el tema pero quise dar mi enfoque personal y diferente para no repetir lo que ya hay.

Me basé en estos pilares básicos para darle forma al tomo

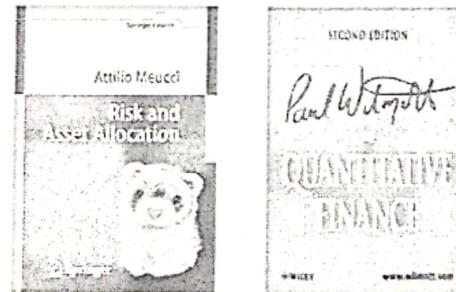
- Ser lo más amplio posible al respecto del tema backtesting, no dejar cosas totalmente afuera
- Ser pragmático y volcar con ejemplos reales y prácticos (aunque simplificados)
- No puse nada de IA / Machine Learning porque va a haber un tomo completo de este tema.

Dicho esto, este tomo es un compendio de herramientas, una simplificación de muchas cosas y sobre todo la bajada a ejemplos prácticos del mercado basados en Python. Tiene una primera parte teórica con más sarasa que código, pero necesaria para darle un marco a todo este tema, y luego sí arrancamos con ejemplos y a puro código. Mi plan original mientras escribo esto es no superar las 200 páginas para que sea un libro llevable, veremos que sale.

No obstante, para quienes quieran mayor profundidad y desarrollos más técnicos les voy a dejar algunas referencias bibliográficas para ampliar conceptos, me tomé el trabajo de filtrarles un amplio espectro desde algunos más conceptuales hasta los más técnicos para que tengan para elegir en función de lo que busquen profundizar, recomiendo fuertemente que antes de ponerse a estudiar esta bibliografía tengan muy claras las herramientas de estadísticas que expliqué en el t[6] de esta colección de Python para finanzas Quant.

De los siguientes libros no es que recomiende ninguno en particular, y mucho menos hacen falta para seguir esta obra, simplemente son referencias para ampliar un poco, de todos se saca algo diferente y la idea de dejarles esas referencias es porque hay muchas cosas que me hubiera gustado ampliar en este tomo y me quedaron afuera.

Hay libros referentes en cuanto a profundidad pero ya son para un nivel más técnico, como por ejemplo "*Risk and Asset Allocation*" de **Attilio Meucci** o la gran obra "*Paul Wilmott On Quantitative Finance*" obviamente de **Paul Wilmott** que son 3 tomos insuperables en calidad de contenido, pero como les decía ya son material más apuntado a un nivel más avanzado.



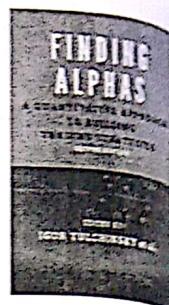
Volviendo a obras más terrenales para todo público de donde se puede ampliar en muchas cosas relacionadas al backtesting, les dejo estas reseñas:

FINDING ALPHAS

A Quantitative Approach to Building Trading Strategies

Autor: Igor Tulchinsky

Es un libro más conceptual que práctico medio básico, pero ayuda a ir adentrándose a cuestiones como el Alpha de una metodología sistemática, los arbitrajes estadísticos, los bias, el overfitting, factores de riesgo, algún concepto de machine learning y muchas ideas de aplicación de backtests sobre ideas de análisis fundamental mas que de indicadores basados solo en precio.



Backtesting Value at Risk and Expected Shortfall

Autora: Simona Roccioletti

Hace Backtests de indicadores de riesgo como VaR y backtests en gral por indicadores de riesgo. Análisis empírico con estimaciones de distribución de variable continua y kermels de densidad



Handbook in Monte Carlo Simulation

Applications in Financial Engineering, Risk Management, and Economics

Autor: Paolo Brandimarte

Muy llevable y completa base estadística que complementa muy bien el t[6] de esta colección un poco más técnico y con código en R y Matlab. Es más teórico y menos aplicado, un poco denso y poco práctico pero muy completo y bien desarrollado.



Learn Algorithmic Trading

Build and deploy algorithmic trading systems and strategies using Python and advanced data analysis

Autores: Sébastien Donadio y Sourav Ghosh

Proporciona un framework de trabajo para Backtests, si bien los ejemplos son demasiado básicos, tiene mucho código Python y ejemplos bien prácticos, un opuesto de la reseña anterior.



Mathematics and Statistics for Financial Risk Management

Autor: Michael B. Miller

Muy pero muy sencillo repaso por la base matemática necesaria para libros más técnicos de la temática quant, está muy piola y fácil para un nivel pre-universitario con ejercicios y respuestas a cada tema. Este lo cito porque me han preguntado muchas veces por libros de matemática básica para finanzas quant.



Que es el Backtesting

Un backtest es la recopilación y posterior estudio de los hipotéticos trades de una estrategia de trading automatizada previamente definida, para saber cómo le hubiera ido en el mercado a esa estrategia de haberla aplicado

No es exactamente una simulación ya que puede parecerse más a una "suposición" porque esta suposición va a asumir muchas cosas que luego en la práctica no tienen por qué ser así, mientras que una simulación debería intentar simular lo más parecido posible las situaciones a la práctica.

Luego una vez asumidas las situaciones y los "supuestos" trades realizados por nuestro bot, se procede a medir los resultados, a hacer un análisis de sensibilidad de las variables involucradas y a plantear esquemas de portabilidad del modelo, es decir, es una tarea con varias etapas, de hecho, algunas más que las mencionadas como el data-feed, el data-profiling, data-mining etc pero ya veremos más exhaustivamente cada una de ellas más adelante.

Algoritmos backtesteables y no backtesteables

Antes que nada, hay que diferenciar distintos tipos de estrategias entre las que son y no son backtesteables, pero mejor para ello primero hagamos una listita de diferentes tipos de bots o de algoritmos de trading automatizado que podemos ver en el mercado y luego con un panorama más amplio veamos cuales se pueden backtestear y cuáles no, pero sobre todo analizaremos el "Por qué"

Tipos de Bots de trading automático

- A- Bots de Colocación de grandes órdenes, para no alterar precio
- B- Bots de Colocación de órdenes, para obtener el mejor precio
- C- Bots de Market Making
- D- Bots de Arbitrajes inmediatos (entre mercados, mismo instrumento)
- E- Bots de Arbitrajes no inmediatos, en un mercado, con sintéticos
- F- Bots de Arbitrajes estadísticos (en un mercado o entre mercados)
- G- Bots de estrategias de opciones
- H- Bots de Balanceo de carteras (optimización, riesgo, etc)
- I- Bots de ruteo para seguimiento de índices (fondos, etfs)
- J- Bots de swing trading
- K- Bots de scalping

Puede subdividirse un poco más, o podría también agruparse en algunas categorías menos, esta es solo una organización que a mi criterio me parece muy razonable. Pero veamos mejor un repaso de cada tipo de automatización para entender mejor cuando aplica o no un backtest y por qué.

Empecemos viendo uno por uno estos bots, para no quedarse solo con el típico bot de profit que es aquel direccional que apuesta a comprar "barato" y vender "caro" más tarde.

A- Bots de Colocación de órdenes grandes

Esto lo habrán escuchado muchas veces como bots de órdenes iceberg.

Inputs del bot:

- Cantidad a operar
- Instrumento
- Lado (compra o venta)
- Precio mínimo en ventas o máximo en compras (puede ser un % del precio de mercado)
- Cantidad mínima (puede ser monto mínimo a operar)
- Plazo máximo

Objetivo: Lograr colocar la totalidad de la orden o el mayor porcentaje posible dentro del rango de precios estipulado, sin mover el precio de mercado más allá del input solicitado

Es backtestable: No

Ojo, antes que nada, vamos a relativizar un poco esto, seguramente lo primero que les viene a la mente en este tipo de bots es BlackRock operando miles de millones en AAPL, y está bien es un ejemplo correcto, pero también podrían ustedes mismos necesitar este tipo de bots para sus inversiones personales... Es que lo de "orden gigante" es relativo a la liquidez del instrumento, mis compatriotas argentinos me comprenderán perfectamente si quisieron operar alguna vez acciones del panel general, porque a lo mejor una simple orden de usd 10.000 que bien podría ser de un particular, para instrumentos muy ilíquidos es un montón

Ahora, esto de saber cuándo estamos hablando de una orden iceberg o no, es decir a partir de qué monto/liquidez decimos que tenemos una orden iceberg, no es tan sencillo, ¿Por qué? Porque hay liquidez oculta, hay sintéticos y coberturas temporales hasta cerrar el hedge, y muchas cosas, mas, pero para simplificar podemos estudiar el tema relacionando 3 variables:

- Monto a colocar
- Volumen medio del instrumento
- Intervalo de tiempo

Claramente si tenemos que colocar una orden por el 5% de lo que mueve un instrumento en ese plazo, no vamos a afectar mucho el precio, pero si ese % en el mismo plazo es el 50% seguramente moveremos el precio bastante. No hay un % estándar porque depende de la dinámica del mercado, hay veces que un % enorme pasa desapercibido sin mover mucho el precio porque en ese momento el mercado esta con muy buen volumen en general más allá de nuestro instrumento y los bots de ratios y arbitrajes estadísticos ayudan al objetivo de no mover el precio

Los bots de este tipo deben ante todo evitar "ser leídos", es decir no tienen que ser obvios, deben ser más pasivos que activos (es decir, que la mayor cantidad de operaciones deben colocarse entre el bid y el ask, sin quitar liquidez del libro "pegándole" a la otra punta).

Dicho esto, ¿es backtesteable un bot de este tipo? Claramente no, porque el hecho de ir poniendo las órdenes va a mover el mercado, y esa reacción justamente depende del mercado, no se puede simular sin una contraparte real, ahora, en determinadas condiciones (plazos largos) donde no juegan un rol tan importante los market makers, puede recrearse una simulación decente para estimar al menos que tanto podríamos afectar, pero ya no se trata de un backtesting, sino más bien de una simulación bastante compleja para los alcances de este libro.

B- Bots de Colocación de órdenes para obtener mejor precio

Este tipo es similar al anterior, solo que en este caso el limitante no es el "Q" es decir la cantidad, sino que pasa a ser el precio, es decir imaginense que les dicen algo así:

"compra todo lo que puedas de GGAL hasta USD 8.2 antes del cierre"

Ahí cambia que no tenemos la obligación de comprar una cantidad "Q" de nominales (obviamente algún tope nos ponen, pero en principio es como si fuera infinito) es decir, el limitante es no sobrepasar el precio, en cambio en el ejemplo anterior tenía que operar si o si una cantidad Q

La diferencia sustancial es que en el caso anterior, si no se operaba una cantidad "Q" tenía que al vencerse el plazo ir a mercado al precio que sea, con lo cual eso se transformaba en algo medio inbacktesteable porque tiene una alta sensibilidad a la liquidez al fin del período

En cambio, este segundo tipo es un poco mas "continuo" es decir se puede desfragmentar todo el período o plazo e integrar para obtener un resultado estimado.

Por lo tanto, este tipo de bot es un poco más simulable que el anterior, generalmente con una buena simulación de Montecarlo y partiendo de un buen fiteo de distribución de los valores del libro de compras y ventas se puede simular en forma aceptable el resultado de la esperanza matemática de la cantidad operada al precio dado

Ahora vean la sutil diferencia, dije "es mas simulable" y no "es mas backtesteable", esto se debe a que en realidad la manera de probar este tipo de bot antes de mandarlo a la cancha es una simulación y no un backtest, son cosas diferentes, en estas simulaciones vamos a tener un output del tipo:

"La cantidad posible a comprar de GGAL a un precio hasta USD 8.2 antes del cierre dado el precio actual y el tiempo que queda es de 150.000 nominales con un desvío estándar del 20%"

Claramente mas que un backtest, es un Montecarlo sumado a un análisis estadístico, ya vamos a realizar estas simulaciones en el tomo de Bots, lo importante por ahora era diferenciar un backtest de una simulación.

C- Bots de Market Making

Este tipo de bot es un continuo, es decir, que no tiene plazo de finalización ni objetivo de finalización, sino que es un bot que está indefinidamente ahí operando

Inputs del bot:

- Instrumento
- Datos paramétricos del mismo (horarios, spreads y volúmenes medios e históricos)
- Share de participación objetivo

Objetivo:

El objetivo de estos bots es venderle a todo sujeto que se meta a comprar ese instrumento y comprarle a todo sujeto que se meta a vender ese instrumento (obviamente que el bot nunca va a lograr el objetivo al 100%, pero la idea es acercarse a ello lo mas posible, con lo cual, de lograrse ese 100% utópico que es lo que ponemos como 3er input, el bot se quedaría con el spread*volumen de todo lo operado en dicho instrumento).

Que hace el bot? Bueno, simple se pone siempre a tapar el bid y el ask, pero también juega con muchas otras variables como ser "precio arbitrado de un sintético" o "precio del mismo instrumento en otro mercado referente" la idea es ser siempre "pasivo" es decir que siempre le lleven su oferta en el ask o le den su oferta en el bid, pero nunca agredir a ninguna de las puntas, bien calibrado un bot de este tipo termina neutro, es decir la cantidad de ventas se equilibra con las compras, en el medio puede quedar expuesto direccionalmente al subyacente que está comprado o vendido temporalmente, y es el riesgo que asume el bot

Dicho esto, que les parece? Es backtesteable o no?

Bueno, claramente no lo es, simplemente porque afecta directamente la microestructura del mercado, por lo tanto, la única manera de saber en cada operación si se hubiera ejecutado o no, es en cada momento estar allí adentro del mercado operando

Y es una simulación mucho más compleja que las otras anteriores, porque en las anteriores, podríamos tomar suposiciones como "el precio de ejecución será el precio medio entre bid y ask en cada momento", y sería un supuesto aceptable para reducir la complejidad de la simulación, pero en este caso del bot de market making no podemos asumir eso porque justamente la gracia del bot es quedarse con la mayor proporción posible de ese spread, de hecho, ese 3er input que puse es "LA" variable a parametrizar, porque a mayor pretensión de spread, posiblemente el bot opere poco y se lleve mucho mas precio un bot competidor, en el otro extremo, a menor spread pretendido puedo tapar a los bots competidores.

D- Bots de arbitrajes inmediatos

Inputs del bot:

- Instrumento
- Mercados
- Costo transaccional en ambos mercados
- Costo transaccional fijo de infra tecnológica prorrataeado
- Desarbitraje mínimo para operar
- MetaDatos (Ej: id del mercado referente, outliers)
- Límites de Volúmen operables

Objetivo: El objetivo de este tipo de bot es muy sencillo, la idea es comprar una cantidad Q del activo X en un mercado M a un precio P , y vender una cantidad Q_2 en un mercado M_2 a un precio P_2 de modo que:

$$P < P_2 \quad \text{y} \quad Q = Q_2$$

Obvio que si $P_2 > P$, entonces se hace igual el trade pero se vende en M y se compra en M_2

Siendo la ganancia del trade:

$$Q |P - P_2|$$

Obvio que si $P > P_2$, entonces se hace el trade pero se vende en M y se compra en M_2

La idea es que esto sea lo mas instantáneo posible, cosa que cerrar la operación antes de que alguna de los dos mercados se mueva

Es backtestable este tipo de bot?

Si, pero... ¿pero qué?

Antes que nada veamos que sería backtestear este tipo de bot

Supongamos que ya sabemos que vamos a arbitrar, tenemos fondeadas las cuentas en exchanges o brokers de ambos mercados, tenemos hedgeada la posición comprada en caso de no poder shortear el instrumento en alguno de los mercados, etc, lo único que nos queda hacer es barrer históricos recientes de ambos mercados, y buscar todos los momentos en que hayan estado desarbitrados de modo que:

$$\text{ASK}_{MERCADO_1} + \text{CostoTransaccionalTOTAL} < \text{BID}_{MERCADO_2}$$

O bien si basamos la operatoria en un parámetro de desarbitraje mínimo operable:

$$\text{ASK}_{MERCADO_1} - \text{BID}_{MERCADO_2} < \text{Desarbitraje_Mínimo_Operable}$$

En cada momento que se cumple esa condición, simplemente compramos en M_1 y vendemos en M_2 a mercado pegándole a las puntas en cantidad igual a la menor de ambas puntas y listo, de seguir

habiendo arbitraje con la punta siguiente en alguna se sigue iterando el proceso. Al final se suman todos los supuestos trades con posibilidad de arbitraje y ya, tan simple como eso se sabría exactamente cuánto hubiera dado de ganancia el bot en ese período.

Bueno, en realidad (y se los digo por basta experiencia propia en el campo) los backtests de bots de este tipo fallan rotundamente, pero rotundamente cuando en alguno de los dos mercados involucrados hay otro bot arbitrador ¿Por qué?

Bueno por competencia de bots, acá juega un rol fundamental el seteo del "costo transaccional" y del "desarbitraje mínimo para operar", imaginen lo siguiente, supongamos que hay dos mercados con el mismo instrumento en donde el backtest da altísimas posibilidades de arbitraje en determinados momentos, pensamos que está libre de competencia y nos mandamos con nuestro bot cuyo backtest dio un resultado excelente.

Al principio nuestro bot con un costo transaccional total de 0.6% y un desarbitraje mínimo para operar de ponele 0.8%, empieza a andar de maravillas, nos empezamos a frotar las manos, pero resulta que en realidad ya había otro bot antes, solo que tenía seteado el costo transaccional en ponele 1% y el desarbitraje mínimo para operar en 1.2%, claro, cuando hay un desarbitraje ponele de 1%, el otro bot no hacía nada y nuestro backtest daba de maravilla diciéndonos que íbamos a ganar 0.4% un montón de veces al día (1%-0.6%, o sea el desarbitraje menos nuestro costo transaccional total).

Esto es así pero hasta que se aviva el del otro bot, y hace otro backtest y le dan muchas posibilidades de arbitrajes pero mas que nada todas cerca de 0.8% (que es cuando actúa nuestro bot que está seteado con ese porcentaje para operar), el tipo se aviva que vino competencia a su bot de arbitrajes en los mismos mercados y se pone a afinar el lápiz, se da cuenta que si yo llegue a un costo transaccional + ganancia = 0.8% el también puede bajar su pretensión de ganancia y bajar el costo transaccional, tarde o temprano logra alguna de las dos, con lo cual seteará el "desarbitraje mínimo para operar" debajo de 0.8% y mi bot quedará pagando y no operará mas nada porque se queda esperando oportunidades de desarbitraje $\geq 0.8\%$ y nunca pasa porque el bot de la competencia seteó su bot en 0.7% y nunca llega a 0.8%

Y así, luego me avivo yo y si puedo sigo bajando ese % hasta que uno de los dos se da por vencido y se estabiliza el % de desarbitraje operable (hasta tanto no entre un nuevo competidor)

Por lo tanto el backtest inicial no tiene sentido, más que para simplemente, saber donde está seteado el porcentaje del mejor bot en el mercado en cuestión. Luego se transformará en una competencia de costos de intercambio (comisiones) y costos operativos (infraestructura cloud), pretensiones y habilidad tecnológica en cuanto a velocidad de operación etc.

En este tipo de competencia, primero se compite por precio y llegado a un punto muy fino donde la volatilidad del tick_by_tick es mas alta que la ganancia neta de cada trade, se transforma en una competencia de velocidad de lectura y ejecución de los trades.

E- Bots de arbitrajes no inmediatos

Este tipo de bot es en su estructura lógica muy similar al caso anterior de arbitraje inmediato pero un poco más compleja la instrumentación, dado que no se da con dos activos exactamente iguales sino con sintéticos, solo por poner un ejemplo de arbitraje de tasas muy típico:

- Se toma prestado un monto M a una tasa " r " por un tiempo T
 - Se compra una cantidad " q " del spot de un activo a un precio " S ", de modo que $M = S * q$
 - Se vende el futuro de un activo a un precio $VF = S * (1+i)^T$ (siendo " i " la tasa implícita del futuro)
- Si se cumple que $i > r$ (la tasa implícita del futuro es mayor a la tasa del préstamo tomado)

⇒ Al finalizar " T ", tendré una ganancia P :

$$P = M * (1+i-r)^T$$

Básicamente es la misma idea que un arbitraje inmediato, pero en este caso lo que se hace es "calzar" todas las operaciones en el momento pero quedan todas abiertas y la ganancia se consuma una vez terminado el período, con lo cual la operación depende que pase un tiempo T para hacerse realmente efectiva pero toda la ganancia queda perfectamente determinada al momento de calzar todo cuando se ejecuta en el instante inicial.

Obviamente que para que esto sea posible la tasa tomadora, es decir la del préstamo que tomo para hacer la operación debe ser estable y no variable (hago la aclaración porque hay oportunidades inmensas en el mundo crypto en el sector DeFi pero son tasas variables, con lo cual no se puede usar para arbitrar sintéticos).

Todo el resto caben las mismas conclusiones que para los arbitrajes inmediatos solo que es más compleja la instrumentación y por lo tanto hay mucha menos competencia y además tiene la ventaja de tener infinidad de variantes de posibilidades y combinaciones con lo que siempre deja oportunidades explotables, siempre.

Yendo al tema backtesting ¿Qué piensan? Es backtesteable?

Si, claro que si, y en este caso es más útil que en el caso de los arbitrajes inmediatos porque bien instrumentado el backtest, y bien diseñada la arquitectura del bot, se pueden tener parametrizados muchos mercados y sintéticos equivalentes y hacer bots que funcionan de maravilla durante períodos mas que envidiables, deja realmente ganancias interesantes hacer este tipo de backtest pero, la ingeniería financiera detrás del research de mercados y sintéticos, sumada a la ingeniería tecnológica detrás de los backtests, y la implementación tienen un laburo (costo y tiempo) inicial bastante importante, mas para equipos e instituciones que para particulares diría yo.

F- Bots de arbitrajes estadísticos

Estos personalmente me encantan y los he operado mucho, este tipo de operatoria sin bien les dicen "arbitrajes" en realidad son demasiado sucios para ser llamados así, también se les dice arbitrajes de ratios, el punto es que parten de una suposición de reversión a la media que si bien tiene su lógica no tiene para nada asegurado el hecho que sea algo con reversión a la media.

Partamos desde lo más burdo y lo vamos afinando luego, supongamos el precio de dos bancos del mismo país, con un valor histórico de sigma (volatilidad) similar. A priori podríamos asumir que el riesgo de ambos instrumentos es similar ya que operan en la misma industria (bancos), mismo país y tienen de hecho una volatilidad histórica muy parecida, supongamos también podríamos agregar por ejemplo que el Rolling Pearson o cualquier otro factor de correlación entre los retornos de ambos se mantiene en una franja de 0.6 a 0.8, bastante alto, bien..

Con estos datos, podrías suponer que el ratio de las tasas de las ON o deuda corporativa de ambos bancos oscilará en torno a 1, podríamos hacer un ratio de flujo de fondos de una cartera invertida en un banco respecto a mismo monto en el otro banco, o más jugado directamente podríamos hacer un ratio de precios spot BancoA/BancoB, esto último es lo más común y simple, dado que ambos retornos están muy correlacionados, el riesgo país es el mismo porque operan en el mismo país y el riesgo industria también, entonces podríamos asumir que el ratio de precios de ambos va a oscilar dentro de una franja que parece tipo "estacionalizada" con oscilaciones muy llenas de ruido blanco (azar) pero que en términos prácticos veremos un gráfico de dicho ratio similar a algo así:



En el ejemplo vemos el ratio de precios GGAL/BMA dos bancos argentinos durante un período de 2 meses. Claramente en este caso comprar el ratio sería comprar GGAL y vender (shortear) BMA, y vender el ratio sería vender GGAL y comprar BMA.

La idea de esto es buscar ratios con mean-reversion, es decir que oscilen alrededor de una media, o sea visto en términos chartistas, una lateralización bien marcada. ¿Para qué? Y fácil, para poder fijar una zona clara de compra y una zona clara de venta, y de última usar stop loss, o zonas donde asumimos que hubo una perturbación en alguna variable puntual de alguna de las dos que nos sacó del supuesto de "riesgo similar" o sea que le genera "una deriva" o tendencia al ratio. Esta divergencia sucede cuando claramente el mercado interpreta que uno de los dos bancos tendrá claramente un mejor flujo de ganancias futuras, y por ende lo pricea rápidamente haciendo que el ratio empiece a tomar una forma tipo tendencial que no nos sirve más para nuestro bot hasta que vuelva a estabilizarse dentro de un rango.

Como decíamos un banco puede tener un balance muy bueno mientras que el otro uno muy mediocre, o surgir alguna noticia que afecte mucho mas a uno que al otro, en fin puede pasar de todo, y cada cosa que comparemos que son diferentes en ambos subyacentes agrega un potencial factor de divergencia del ratio, es decir que se salga de la zona de lateralización estable y nos genere una pérdida por tener que cerrar el trade fuera de la zona de lateralización.

Entonces, vienen los ajustes, por ejemplo, al ratio P_A/P_B (precio del banco A sobre precio del banco B), por ejemplo lo podríamos afectar por su EPS (ganancias por acción del último balance, o proyectados del período presente) si llamamos a ese parámetro de corrección: lambda " λ " y lo dividimos por el valor medio en un determinado período de ese mismo, obtendríamos un factor que representa la cantidad de veces que el EPS actual entra en el EPS medio, pero en realidad para normalizar el precio por el EPS nos interesa la inversa de esa división, ya que a mayor expectativa de EPS, mayor expectativa del precio del subyacente, y nosotros lo que queríamos es un ratio estabilizado por los factores.

Así que podríamos re-escribir el ratio "R" ajustado por un parámetro lambda (respecto a la media histórica de dicho valor de lambda) de la siguiente forma:

$$R = \frac{P_A \cdot \overline{\lambda_A}}{P_B \cdot \overline{\lambda_B}}$$

Y generalizando, si tenemos una serie de "n" parámetros lambda cuya variación positiva del mismo se asume directamente proporcional a la futura variación positiva del precio del subyacente, nos queda:

$$R = \frac{P_A}{P_B} \cdot \prod_{i=1}^n \frac{\overline{\lambda_A}}{\lambda_{A_i}} \cdot \frac{\lambda_{B_n}}{\overline{\lambda_B}}$$

Con lo cual nos queda el ratio de precios ajustado por toda la serie de parámetros lambda de ambos subyacentes, a todo esto, no la quiero complicar mucho, pero si los parámetros "lambda" tuvieran varianza muy diferente entre ellos, podríamos ajustar también por las varianza de los parámetros, o usar un zscore de los mismos, en fin, esto deja mucha tela para cortar pero no es el objetivo de este tomo, seguramente en el tomo de bots veremos ejemplos mas completos de este tipo de bot, pero por ahora lo que nos preocupaba era hacer una intro al backtesting de los diferentes modelos de trading y yo quería dejar esta intro tan "así nomás"

¿Cuáles son los riesgos de este tipo de "arbitraje de ratios"?

Simple, que haya algún parámetro "lambda" que no contemplamos en el modelo.

Un caso muy pero muy conocido es el del fondo LCTM (Long-Term Capital Management) que contaba con cerebros como Myron Scholes y Robert C. Merton, nada menos (ambos premios nobel de economía por el modelo conocido como Black&Scholes)

Contado muy resumidamente, usaron este tipo de arbitrajes de ratios apalancados, estaban tan confiados en su modelo estadístico (ojo, tenían con que, eran terribles cracks, pero bueno, se ve que poco humildes) que ni siquiera pensaron en la mínima remota posibilidad de que se les haya escapado algún lambda que los exponga demasiado si se iban mucho de las zonas predefinidas para los ratios, cuestión que entre el apalancamiento y la necesidad a asumir pérdidas tempranamente y eso sumado a la falta de liquidez al querer salir (porque eran bastante grandes en los mercados que operaban) hizo todo esto que entró en un círculo en donde ya los socios desconfiados al retirar capital les jugaban en contra porque aumentaba más el ratio de apalancamiento al disminuir sus activos, y los agarró la crisis asiática de 1998 y el default ruso en el mismo contexto y caput, tuvieron que ser rescatados para que no generen una crisis financiera general.

Bueno, volvamos a lo nuestro, ¿el arbitraje estadístico o arbitraje de ratios, es backtestable?

Si, claramente, y es de lo mas backtestable en estrategias de trading automatizado.

Ahora el problema es que suponen esquemas bastante complejos en comparación con otros tipos de backtest más direccional que veremos más adelante, de hecho hay libros enteros de arbitrajes de ratios como el de A. Pole que es un libro muy escueto y ni habla de backtesting pero sí de varios aspectos a considerar para un modelar arbitrajes de ratios y son unas 250 páginas, solo de marco conceptual, así que imaginen el laburo que puede llevar diseñar una buena arquitectura de backtesting de este tipo de modelos.. Bueno, hecho ese disclaimer les voy a dejar al final del libro como un apéndice de otros tipos de backtest, un ejemplo de como backtestear ciertos parámetros de los supuestos, no así el modelo en si, o el backtest final del bot, sino de los supuestos principales.

G- Bots con opciones

Acá hay varios tipos, por ejemplo:

- Bots de Hedging
- Bots de estrategias de spreads
- Bots de estrategias tipo delta neutral, o basados en vega o theta
- Bots direccionales

Como se imaginará la complejidad de las opciones da para mucho, de hecho, va a haber un tomo entero dedicado a opciones, por lo que no me voy a meter por ahora mucho con este tema.

En el caso de bots con opciones además del parámetro "precio" que es "EL" parámetro independiente "x" de la $f(x)$ que me devuelve un "y" a backtestear, aquí en opciones tengo múltiples parámetros

independientes de los cuales depende mi resultado de $f(x)$ a backtestear, entre ellos, la volatilidad, el tiempo, la tasa, los dividendos, etc.

Dicho esto, backtestear una función que depende de varios parámetros es ni mas ni menos que backtestear cada parámetro en su marco teórico correspondiente. Luego se sacan conclusiones parciales y en muchos casos se usan estas conclusiones como hipótesis de entrada para un backtest final, por ejemplo, supongamos que vamos a tradear una estrategia de Long-Strangles, es una estrategia que inicialmente es delta-neutral y es muy dependiente de vega o la sensibilidad a la volatilidad, con lo cual vamos a hacer mucho foco en un backtest de la volatilidad, es decir de HV y de IV, las volatilidades históricas del subyacente y otro backtest de las volatilidades implícitas históricas y presentes de sus opciones.

Una vez hecho esto, con esos resultados vamos a tomarlos de hipótesis como parámetros de entrada fijos o dinámicos en nuestro modelo de backtest del resultado de una $f(x)$ donde "x" ahí ya es el precio de un call +un put, que sabemos dependen de su volatilidad histórica que a su vez está representada por una volatilidad implícita, por lo que al backtestear vamos a suponer una cierta relación entre ambas que habrá salido de un previo backtest, es decir **es como una mamushka de backtests**.

¿O sea que es backtesteable?

Si claro que sí, pero son como varios backtests dependientes.

H- Bots de Balanceo de carteras (optimización, risk-management)

Inputs:

- Espectro de activos elegibles
- Criterio de filtros dentro de los elegibles para incluir en la cartera
- Restricciones (% de ponderación mínimos y máximos, combinaciones, etc)
- Frecuencia de actualización y ventana de tiempo (hacia atrás)
- Racional de riesgo

Objetivo: Rebalancear periódicamente una composición de cartera de modo tal que el algoritmo maximice la relación retornoEsperado / riesgoAsumido

Esto es debido a que asumimos que un potencial retorno más elevado conlleva el hecho que está asociado a un mayor riesgo, generalmente se mide el riesgo en función de la volatilidad armando ratios como el Sharpe Ratio, o el Sortino, pero también hay otras medidas de riesgo como el riesgo de cola o riesgos de drawdowns, etc.

La idea es entonces, dado un racional de riesgo (definir qué es lo que asumimos como medida de riesgo), backtestear los diferentes algoritmos, con diferentes métricas y parámetros de condiciones que me generen un resultado óptimo final de la cartera, esto no significa encontrar la cartera de mayor riesgo esperado, sino la cartera de mejor relación retorno/riesgo, vamos a tomar como primer modelo

un ejemplo sencillo basado en la MPT (Teoría moderna de portafolios de Markowitz) y luego le daremos alguna vuelta de tuerca y abriremos el esquema para que cada uno pueda profundizar en este tema de backtest.

Como sería el flujo de acción del algoritmo en que rebalancea:

1. Buscar una serie de activos elegibles usando algún tipo de criterio de filtro
2. Buscar en una ventana de tiempo dada, la ponderación ideal de los componentes elegidos
3. Calcular la diferencia entre ponderación P en P_{t-1} y P_t
4. Rebalancear cada activo de la cartera

¿Es backtestable este tipo de bot?

Por supuesto, de hecho, este es uno de los tipos de bot más backtestable que hay, y la idea es en el backtest encontrar los parámetros óptimos, estos parámetros son la frecuencia, la ventana a considerar hacia atrás (punto 2 del flujo de acciones), las restricciones de % mínimos y máximos de cada activo, los activos iniciales, el criterio de preselección etc.

En definitiva, la idea general es basarse en el concepto de la diversificación y reducción de la varianza total para minimizar el riesgo para un mismo retorno, o por el contrario, maximizar el retorno para un mismo nivel de riesgo (volatilidad) asumido.

PROs:

- Es un campo muy estudiado y demandado por los administradores de carteras
- Es muy backtestable y genera buenos resultados en la mayoría de regímenes de mercado

Contra:

- En eventos de cola, eventos de grandes caídas las correlaciones de todos los asset tienden a 1 por lo tanto no tiene gran efecto toda la diversificación
- Parte de la base de tener que asumir un "retorno esperado" y "volatilidad esperada" lo cual es imposible de conseguir, con lo cual ya parte de estimaciones, pero bueno, siempre es mejor que estar a ciegas sin plan obviamente.

Yo personalmente prefiero el "tail risk management" para poder estar posicionado con una asset allocation mucho más arriesgado, es decir, asumir un riesgo mucho mayor, pero tener una estrategia de hedge (en mi caso con estrategias de opciones) que me deje dormir tranquilo con esa exposición, pero así y todo una vez definido el nivel de riesgo a asumir, siempre suma dentro de ese nivel predefinido, buscar la mejor combinación posible (en los papeles, según datos pasados) de rentabilidadPotencial vs riesgoAsumido.

Bueno, ya veremos más en detalle algunas cosas, si bien este no es un tomo de portfolio management, este tipo de análisis de portfolio es muy útil en combinación con otros backtest, ya que como veremos podemos usar sintéticos de estrategias como "assets" y proyectar un portafolio de este tipo de sintéticos.

I- Bots de ruteo para seguimiento de índices (fondos, etfs)

Este tipo de bot es más pasivo, es decir, solo tiene que "trackear" determinada ponderación de "assets" prefijados, si bien en principio no hay nada que backtestear, en realidad lo que sucede en la práctica es que siempre aparece un "tracking error" es decir un leve desvío entre el índice replicado y nuestra ejecución, esto se debe a que se operan este tipo de bots no solo con los mismos "assets" sino con sintéticos y además las ejecuciones son en tiempo real, tick by tick, con lo cual es imposible que en cada cambio de estado del mercado se haga una re-alocación perfecta de todos los componentes simultáneamente, con lo cual se suelen simplificar las ejecuciones de este tipo de bot a un determinado timeframe y con determinados márgenes de operatoria, todos lambdas o parámetros optimizables y backtesteadables, y es una tarea super interesante desde lo tecnológico pero mas para desarrollar en grandes equipos de desarrolladores, con mucho de patrones de diseño y demás, tema super interesante pero que escapa al alcance de esta obra, por lo que simplemente lo menciono en esta enumeración y les comento que es un tipo de bot muy pero muy estudiado en lo que llamamos el sell-side (las instituciones que justamente ofrecen este tipo de ETFs pasivos).

J- Bots de swing trading

Bueno, por fin llegamos a los bots de profit, que son aquellos que operan una determinada señal de compra, y luego una determinada señal de venta, timeout, stoploss o takeprofit para salir, esperando generar una ganancia entre dichas operaciones que supere al benchmark del mercado.

Este tipo de bot es muy usado en el ámbito del "retail" o sea nosotros los minoristas, y realmente lo vi muy poco en el ámbito institucional, ya que los fondos y bancos de inversión están más ocupados en gestionar los otros tipos de bots que vimos antes.

Inputs:

- Precios en tiempo real y velas anteriores.
- Racional de cálculo de indicadores
- Racional de señal de entrada (indicadores, ej. $\sigma < 50\%$ & $RSI > 65$ & Precio $>$ SMA₂₀₀)
- Gatillo de salida (Señal con indicadores, o timeout, o StopLoss o TakeProfit)
- Constantes de configuración (ej. Costos transaccionales, horarios de mercado etc)
- Otros parámetros de gatillos (capital en la cuenta, volumen en liquidez con x% profundidad, etc)

Objetivo: Como dijimos, el bot debe comprar cuando se den los gatillos de compra y vender cuando se den los de venta, simplemente debe ejecutar el racional predefinido. Generalmente el timeframe que se usa es de velas o precios diarios y, si bien una clasificación exhaustiva podría listar unos 10 tipos fácilmente, vamos a separarlos en 3 tipos bien diferenciados.

Cabe aclarar que a su vez cada uno de estos tipos podrían subdividirse en tipos diferenciados entre si pero a grandes rasgos para no irme por las ramas acá estas serían las categorías principales:

Tipos de bots de swing trading:

- Trend-Following
- Contrarians
- Lateralizadores

Bots Swing-Trading del tipo Trend-Following

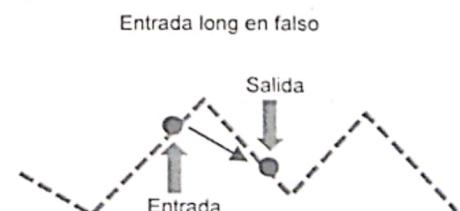
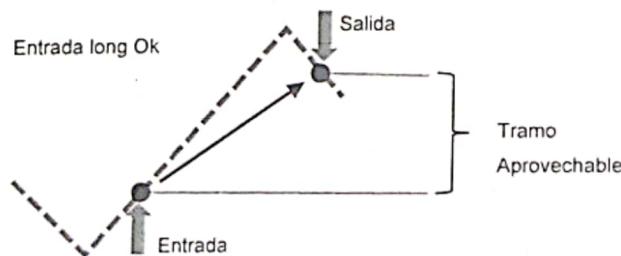
Como su nombre lo indica los del tipo "Trend-Following" son aquellos que siguen señales tardías de una tendencia intentando agarrar las mismas cuando ya se empiezan a desarrollar, e intentan aprovechar la mayor parte posible de dicha tendencia y salen o bien por un determinado take-profit o stop-loss o bien por una nueva señal de salida que indique que se ha perdido la tendencia.

El siguiente es un ejemplo básico a modo didáctico de una señal dada por un simple cruce de medias, la idea de estos bots como se ve es agarrar "una parte" de la tendencia lo mejor posible y salir cuando queda claro que la tendencia está agotada.

Como ven tiene miles de cosas criticables, como por ejemplo que en este caso "sale demasiado tarde" si se usa la misma señal de entrada para la salida. Pero veremos los problemas de estos bots más adelante, justamente en los backtestings.

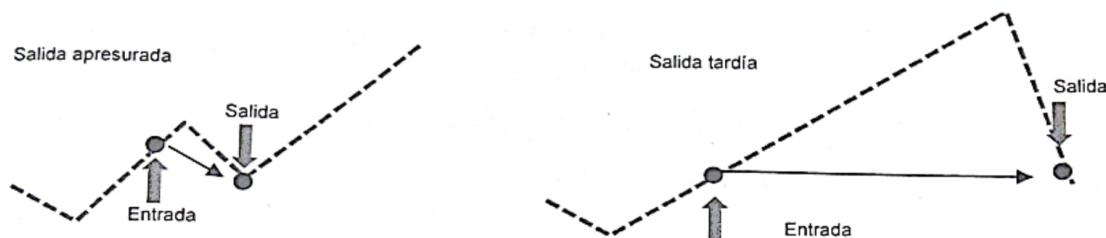


Como ven en los esquemas siguientes en ambos casos, sin saber "la vela de la derecha" al momento de la entrada, pareciera haber comenzado una tendencia que queremos seguir, pero en un caso se desarrolla y en el otro no, es parte de una lateralización. El tema es que si esperamos mucho o una señal muy fuerte para entrar y no comernos amagues, perderíamos gran parte de la parte de las tendencias buenas.



El mismo dilema sucede en las salidas, si ponemos un criterio de salida muy estricto, como en el primer ejemplo, podemos caer en una salida apresurada que nos saca de una buena tendencia solo por una pequeña corrección, por el contrario, si ponemos un criterio demasiado laxo, en esas caídas violentas nos comemos todo el profit de una buena tendencia agarrada.

El primer caso se podría corregir con un segundo gatillo de salida tanto de temporal como de movimiento, y en el segundo caso se podría corregir también con un time-out o un take-profit



Sin embargo, como veremos, una parte de la función de un backtesting, es justamente analizar la sensibilidad de este tipo de parámetros de entrada y salida, al menos para saber si hay "zonas óptimas" o zonas de convergencia de los parámetros que optimicen el resultado del método sin overfitear

Problemas de los Swings Trend Following

- Señales de entrada tardías: Cuando la tendencia es muy corta y arranca con un GAP o unas primeras velas muy fuertes, si luego no se desarrolla una tendencia larga, probablemente por mas que configuremos muy bien la salida, podemos a pagar demasiado peaje al entrar. Puede mejorarse este tipo de señal con un refuerzo o confirmación que mida justamente si las últimas velas no son parte ya de un gran movimiento inusual, a lo que dicha señal o refuerzo haría esperar una corrección a la señal original para activarse.
- Señales de Salida demasiado tardías: A diferencia de las señales de entrada, las de salida cuentan con la ventaja de poder haber aplicado un gatillo de stop-loss o de take-profit antes, pero nos juega en contra que las bajadas suelen ser mucho más fuertes y rápidas que las subidas, con lo cual en muy pocas velas se puede perder todo lo ganado en una linda tendencia agarrada, con lo cual suele ser muy complejo configurar (y backtestear) bien estas señales. En este tipo de ajuste para mejorar la performance por señal de salida, es mucho más útil un sistema de backtest del tipo "Event-Driven" ya lo veremos más adelante, tiene una complejidad computacional mayor, pero es más preciso para medir un desarrollo de un evento dentro de otro. En este caso la desactivación de una tendencia dentro del desarrollo de la misma, por ahora se los comento a modo conceptual, ya veremos esto más adelante.
- Régimen de Lateralización: Como se imaginarán un régimen de lateralización es devastador para este tipo de Bot seguidor de tendencias, ya que estaría todo el tiempo tratando de entrar a una tendencia una vez que se empieza a desarrollar, pero justamente un mercado lateral,

está todo el tiempo "amagando" que va a desarrollar una tendencia y no termina yendo nunca a ningún lado, y eso hace que al entrar tardeamente en una tendencia que no se llega a desarrollar se entra y paga justo en las bandas superiores de la lateralización y se sale en las inferiores dando resultados horribles. Una buena forma de mejorar los bots de trend-following para evitar operar (u operar menos) en mercados laterales es armarse un clasificador con algún modelo de IA del tipo de mercado en el que se viene desarrollando nuestro activo, ya lo veremos en el tomo de IA este ejemplo.

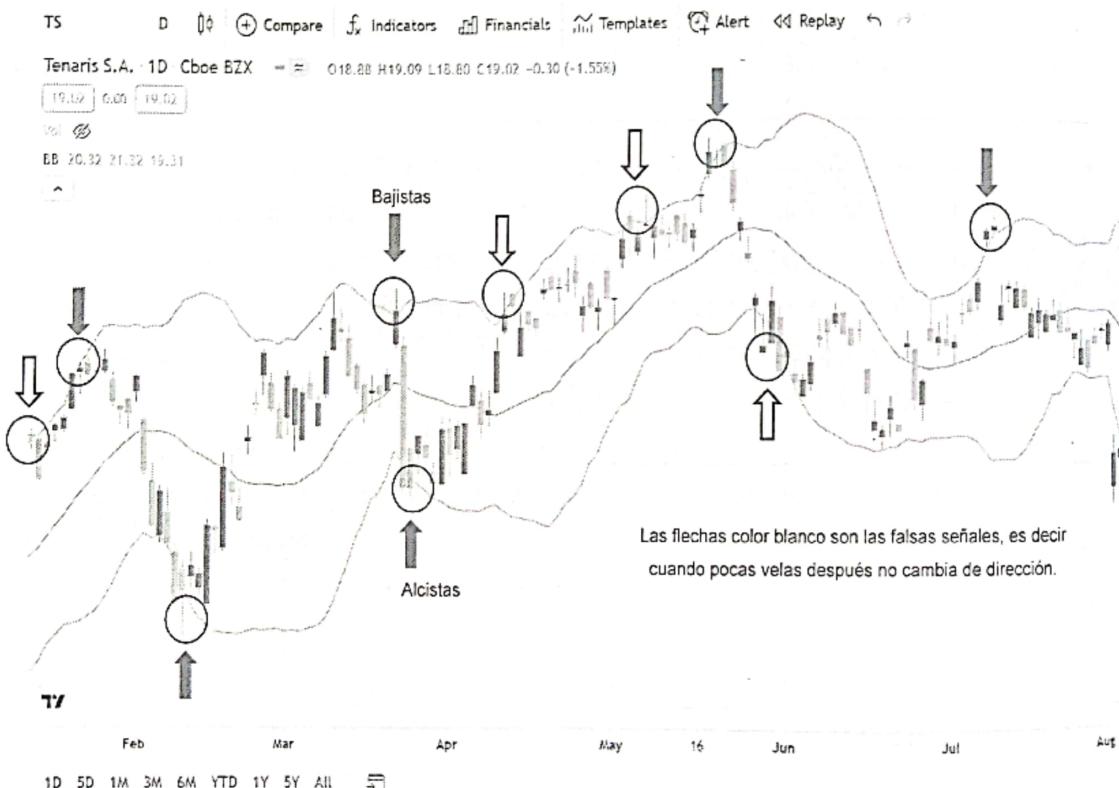
Como anécdota les cuento que cuando empecé a hacer trading algorítmico, uno de mis primeros bots entró en esta trampa y al archivo que disparaba el bot todos los días le puse "sanFilippo.php" :)



Bots Swing-Trading del tipo Contrarians

Este tipo de bots son de muy corto plazo (pocas velas) y buscan anticipar un brusco o rápido cambio de tendencia (repito "anticipar") es decir buscan saturación de una dirección

Por poner un ejemplo simple, un indicador clásico son las famosas bandas de bollinger, que marcan una banda de +/- 2 desvíos estándar de una media móvil, y la "idea contrarian" es que cuando saturan rompiendo una de las dos bandas aumenta la chance de corrección en sentido contrario, es decir cuando pasa la banda de arriba es señal bajista y cuando pasa la banda de abajo es señal alcista (Cabe aclarar que los contrarians son de muy corto plazo)



Como ven las bandas de bollinger por si solas no son para nada un buen indicador contrarian, producen cualquier cantidad de señales falsas, pero lo pongo como idea, quizá combinado con factores de volumen y de señales pasadas termina siendo un indicador aceptable.

Generalmente este tipo de bot tienen un plazo de ejecución muy corto, a contrapelo de los tren-following que busca entrar en largas tendencias, estos contrarians, buscan operar una corrección rápida de pocas velas y salir rápido, por lo que es común que un parámetro típico sea un "n" que refiere a la cantidad de velas para cerrar, o un "TP" takeProfit, etc.

Inputs

- Gatillo de entrada (parámetros de indicadores, etc)
- TimeOut o TakeProfit
- Gatillo de confirmación de salida (opcional)

Objetivo: Como dijimos, la idea es entrar por una señal de entrada que indique saturación del movimiento y salir pocas velas después esperando un cambio de dirección del movimiento previo.

Problemas de los Swings Contrarians

Básicamente se basan en saturación de una dirección, naturalmente esperando una corrección o descompresión de indicadores, esto indica que ya per se, sabemos que apuntamos a momentos extremos para ambos lados, personalmente si estudiara cualquier método contrarian me tomaría muy en serio la diferenciación long vs short porque en momentos de alta volatilidad o extremos son muy diferentes los comportamientos de ambos, pero volviendo al análisis de los problemas de este tipo de bot, bajo mi opinión el principal problema es la sensibilidad a los stop loss. Es decir son métodos que por una pequeña modificación en este parámetro de stop los varía mucho la performance.

Como se imaginarán si setean stop-loss muy cortos, van a quedar afuera del trade asumiendo pérdidas muy seguido, y mas teniendo en cuenta que operar contrarians es operar momentos más volátiles que la media. Por el contrario si ponemos los stops muy largos, pueden aparecer pérdidas muy fuertes si se rompe una lateralización y se dispara para cualquier lado el activo.

Así que en estos casos muy probablemente vayamos a querer backtestear por sobre el resto de parámetros, esta sensibilidad o este punto de fallo digamos.

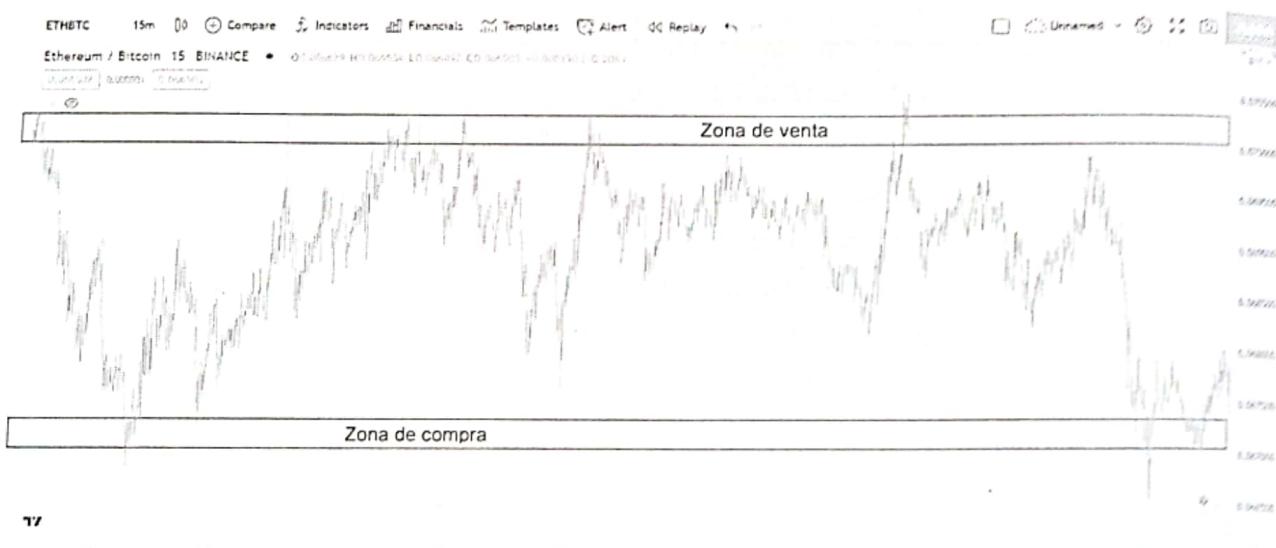
Bots Swing-Trading del tipo Lateralizadores

Este tipo de bot se suele confundir con los bots de arbitrajes de ratios, creo yo porque se construyen con ideas similares aunque no iguales, ya que en los arbitrajes de ratios, como vimos, se buscan activos muy similares o representaciones de precio muy similares o equiparar las diferencias multiplicando por ratios de ajuste para lograr un ratio que oscile alrededor de una media lo mas acotadamente posible.

En cambio en el caso de los swings lateralizadores, comenzamos porque las bandas son móviles, es decir no necesariamente hay siempre una regresión a la media, sino que se operan entre bandas,

Tienen también una similitud a los contrarians en el sentido que se operan contra la tendencia previa, pero aquí en lugar de buscar indicadores saturados, buscamos un activo o ratio muy lateralizador naturalmente, no ajustamos las series, simplemente buscamos activos o régimenes laterales y operamos en bandas.

En el ejemplo mostramos el par ETH/BTC durante 10 días en el mes de Julio de 2021, que como vemos osciló en un rango del 5% entre dos bandas de compra y venta



Un buen punto para buscar activos lateralizadores podría ser buscar pares de activos con buen R2 o simplemente buscar activos que tengan bajo "ratio de información" que es el cociente entre mu y sigma de los retornos.

Problemas de los Swings Laterales

Claramente el principal problema es que uno de los activos del par, o el activo en caso de ser solo un activo lateralizador, se descontrola por algún evento que lo dispare en la dirección opuesta a la que nos falta para cerrar el trade, por lo que en este tipo de bot es fundamental el seteo del stop-loss

Inputs:

- Zona de compra
- Zona de venta
- StopLoss (ambos sentidos)
- Racional de redefinición de zonas de compra y venta

Objetivo: realizar la mayor cantidad de compra+venta en la zona definida antes de cortar por stoploss y redefinir zona de lateralización

K- Bots de scalping

Por último, tenemos bots de scalping que en realidad son los equivalentes a los de swing trading pero en timeframes más chicos, ahora bien, dato clave: Los seguidores de tendencias en timeframes más chicos empiezan a hacer agua más rápido ¿por qué?

Pequeño repaso de Camino Browniano Geométrico

Sin irme por las ramas, si bien hay muchos conceptos y detalles a discutir, básicamente el movimiento de los activos en la bolsa o "sus retornos" tiene 3 componentes:

- 1- Forma de la Distribución: Se suele usar dist normal, pero como vimos en el T[6] fitea mejor una JohnsonSU por ejemplo. Aquí no me refiero a los parámetros que definen la CDF de la distribución sino a la forma en sí, es decir no es lo mismo una variable continua de distribución uniforme que normal o de una Cauchy o una JohnsonSU como decíamos.
- 2- Una deriva dinámica: En el caso de una distribución normal sería el "mu" pero en términos mas generales es el sesgo del movimiento, es decir lo que en bolsa llamamos "la tendencia". Y decimos dinámica porque no es una simple tendencia fija para los 20 años de una serie histórica, sino que se descompone el movimiento en claros momentos bull/bear/laterales y en cada uno de ellos tiene una deriva representativa.
- 3- Ruido Blanco: Es una representación del azar en el movimiento y es el componente principal, que representa en la bolsa, por decir un número, el 80% del movimiento (obviamente por momentos muy marcados de tendencia será menos pero rara vez menor al 70% y por momentos muy laterales será más, superando el 90%)

Básicamente un Movimiento Browniano Geométrico (GBM por sus siglas en inglés) es un proceso de markov, que es ni más ni menos que la composición de estas partes asumiendo una forma de distribución normal y una deriva o "drift" única para todo el movimiento.

Partiendo de la ecuación diferencial en la que S_t es un proceso estocástico que representa un precio de un activo por ejemplo, y W_t es un movimiento browniano, tenemos un primer término que depende de "mu" o la deriva (tendencia) del movimiento que le da la pendiente y un segundo término que agrega la función browniana de ruido blanco aleatorio W_t

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

Y aplicando el lema de Itô (método sencillo para resolver ecuaciones diferenciales estocásticas, llamado así por el matemático japonés Kiyoshi Itô) para resolver la ecuación podemos obtener:

$$\ln \frac{S_t}{S_0} = \left(\mu - \frac{\sigma^2}{2} \right) t + \sigma W_t$$

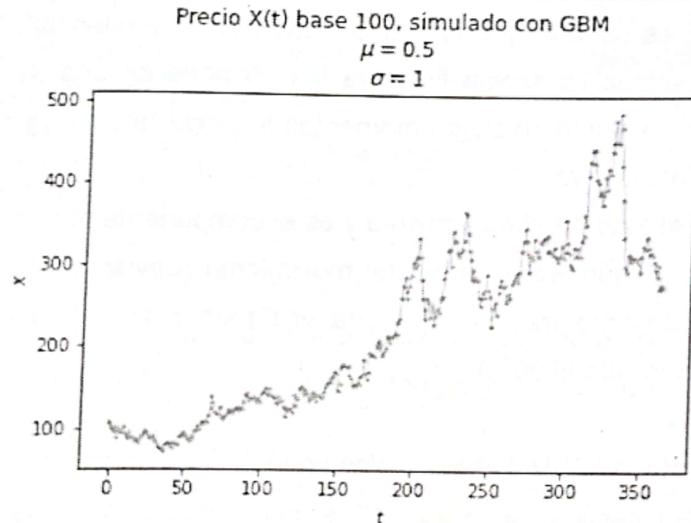
Donde el $\ln(S_t/S_0)$ es el retorno logarítmico del activo para cada momento basados en este movimiento. Aquí también vemos que el primer término de la fórmula marca la deriva mientras que el segundo es ruido que dijimos era un movimiento browniano basado en un ruido normal de $\mu=0$ y $\sigma=1/\sqrt{t}$ (es el mismo concepto de multiplicar por la raíz de t que venimos usando siempre para normalizar la escala de tiempo). Veamos una mini implementación en Python:

```
import numpy as np
import matplotlib.pyplot as plt

mu = 0.5
n = 365
dt = 1/365
x0 = 100
sigma = 1

np.random.seed(1) #fijamos semilla
x = np.exp(
    (mu - sigma ** 2 / 2) * dt
    + sigma * np.random.normal(0, np.sqrt(dt), size=n)
)
x = 100 * x.cumprod()

plt.plot(x, marker='.', markersize=3, lw=0.5, c='gray')
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.title(f"Precio X(t) base 100, simulado con GBM \n $\mu={mu}$ \n $\sigma={sigma}$")
plt.show()
```



Acá simulamos un retorno anual medio del 50% con una volatilidad anual del 100%, y usamos un diferencial de tiempo $dt=1/365$ asumiendo algo que cotice todos los días

Muchas simulaciones de Montecarlo, que es lo que se usa para la parte de simulaciones de los backtests, se suelen implementar asumiendo un GBM pero en el transcurso de este tomo, vamos a algunas implementaciones al menos un poco más sofisticadas, ya que este modelo es demasiado básico, dado que sabemos que en los mercados no se verifican este tipo de movimientos. Pero al menos nos sirve para presentar formalmente el tema y hablar de generalidades del ruido y componente de deriva y desvío en términos generales.

Bien, hecho ese pequeño repaso de lo que es un GBM ahora retomemos con lo que habíamos afirmado acerca del scalping vs el swing-trading:

"La participación de ruido blanco dentro del movimiento aumenta con timeframes más pequeños"

Y esto en términos relativos obviamente, es decir es obvio que en un minuto hay menos oscilación que en un día porque tiene menos tiempo para desarrollar un camino, pero lo que decimos es más amplio, es decir, decimos que el componente de azar en las velas del minuto a minuto es más alto que el componente de azar en las velas diarias, y tiene todo el sentido del mundo ya que justamente lo que cambia es que los eventos que en la bolsa llamamos "drivers" o eventos significativos que sin dudas alteran el precio, se dan muy cada tanto, podría ser una o dos veces por año máximo, con lo cual las derivas suelen tener momentum claros de decenas de días. Ahora si buscamos mini-drivers, se hace todo mucho más confuso, probablemente los haya, pero se diluyen más con el ruido y hacen que la proporción de ruido aumente (ojo estamos hablando igual de proporciones muy elevadas siempre, ruidos de más del 70% del movimiento que queremos estudiar)

Es muy importante tener esto bien claro, no me canso de repetirlo porque justamente en la bolsa tenemos este enfoque que en el resto de las disciplinas es diametralmente opuesto, por ejemplo, cuando estudiamos un movimiento sabremos que el 80% fácil es azar, y vamos a buscar jugar con el otro 20% a favor

¿para qué?

Y porque si podemos de alguna forma predecir un evento binario (sube/baja), solo tomando ese 20% y dejando al azar el otro 80%, vamos a estar con 60% a favor (20% que teníamos mas la mitad del 80% de azar) y 40% en contra (la parte de azar que siempre estará en contra)

¿parece poco 60% a 40%? No, para nada, si pudiéramos operar algo que sistemáticamente me da una probabilidad de acierto del 60%, a la larga y mientras dure, va a ser rentable (con un buen manejo de riesgo y parámetros).

Volviendo a los bots de scalping, como les dije antes, son similares en arquitectura a los bots de swing pero al ser timeframes más chicos, hay mayor componente de ruido, con lo cual por ejemplo los seguidores de tendencia no suelen ser para nada efectivos, al menos yo nunca vi gran cosa en ese campo, pero si en los lateralizadores y en los contrarians, recomiendo igualmente para este campo de scalping enfocarse en un entorno más de IA y algos de machine learning más que en frameworks de backtesting para modelar ideas. Ya lo veremos en el tomo de IA.

Reflexiones Retornos de stocks vs GBM

Ya que en esta instancia tocamos al menos de costado el tema del modelado de los movimientos en la bolsa via un GBM, voy a aprovechar para hacer una mini reflexión acerca de la lógica subyacente en los bots de profit, la mayor crítica de algunos quants a estas ideas es:

"Al ser los retornos un modelo que sigue un GBM es azar puro y no hay nada que backtestear, no se puede ganar consistentemente en este tipo de estrategias"

Obviamente que si estoy escribiendo este libro es porque no estoy de acuerdo con esa afirmación, por varias razones que hacen "LA" diferencia, que si bien es útil, bien explotada, puede ser significativa.

Diferencias en la práctica

1- En primer lugar como dije antes, los retornos en la bolsa no siguen ni ahí un GBM, por un lado la distribución de los retornos no es para nada "gaussiana" si bien en un modelo base puede asumirse una "normalidad" la realidad es que difiere mucho en las colas y en el centro de la distribución, como vimos en el t[6] de esta colección.

2- Lo otro que pasa es que el mercado se rige por eventos, que algunos llamamos "drivers" que van modificando las tendencias de mediano plazo formando como "submovimientos" y el GBM supone un único drift o "mu" para toda una serie

3- En tercer lugar, los mercados se comportan de forma particular en tanto a que cada activo, si bien son empresas diferentes, de sectores diferentes y en países y con clientes diferentes, más allá de todo eso, los activos tienen una "correlación base" como si el mercado en sí tuviera una componente de "mercado" y otra individual para cada activo y el resultado en cada activo fuera un mix de estos dos componentes.

Pero no me crean, verifiquen

Vamos ahora sí al código, vamos a bajar data de las FAANG y calculemos su "mu" y "sigma" y metamos eso luego en una simulación Montecarlo de un camino tipo GBM para los 5 activos durante 500 ruedas, arrancamos por el cálculo del "mu" y "sigma" medio

```
import yfinance as yf

faang = ['AAPL', 'AMZN', 'FB', 'NFLX', 'GOOGL']
data = yf.download(faang, start='2019-01-01', auto_adjust=True)
cierres = data['Close'].iloc[-501:]
cierres.pct_change().mul(100).describe().mean(axis=1)

[*****100*****] 5 of 5 completed

count      500.000000
mean       0.181923
std        2.243488
min       -11.562557
25%        -0.866297
50%        0.156443
75%        1.280768
max        11.248166
dtype: float64
```

Noten que uso el método .mul(100), que como habrán deducido multiplica por 100 la salida del pct_change(), lo que me permite concatenar métodos de forma más pythonica que andar multiplicando con operadores, el mean(axis=1) es porque axis=1 se refiere a las columnas, es decir le pido las medias de los valores del describe() para cada columna (los tickers de las faang)

Una vez que tengo los parámetros ($\mu=0.18\%$ y $\sigma=2.24\%$) simulo las 500 ruedas vía un GBM:

```
import numpy as np
import matplotlib.pyplot as plt

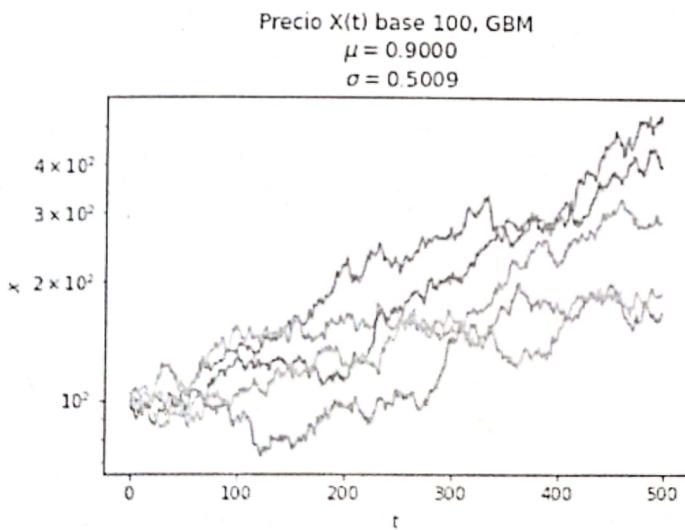
mu = 0.18/100 * 500
n = 500
dt = 1/500
x0 = 100
sigma = 2.24/100 * (500**0.5)

np.random.seed(1) #fijamos semilla

for i in range(5):
    x = np.exp(
        (mu - sigma ** 2 / 2) * dt
        + sigma * np.random.normal(0, np.sqrt(dt), size=n)
    )
    x = 100 * x.cumprod()

plt.plot(x)

plt.yscale('log')
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.title(f"Precio X(t) base 100, GBM \n $\mu={mu:.4f}$ \n $\sigma={sigma:.4f}$")
plt.show()
```

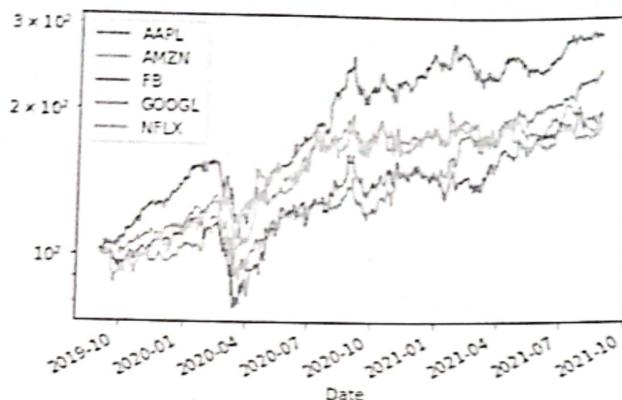


Acá llevamos a un periodo bianual el mu y el sigma asumiendo 250 cotizaciones por año

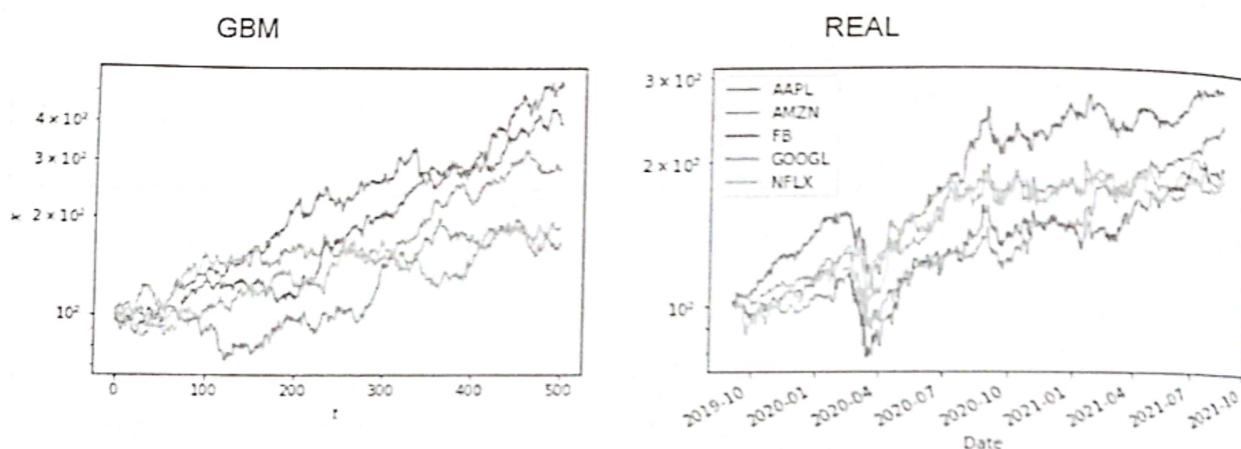
Bien ahora comparemos con el real:

```
import yfinance as yf

faang = ['AAPL','AMZN','FB','NFLX','GOOGL']
data = yf.download(faang, start='2019-01-01', auto_adjust=True, progress=False)
cierres = data['Close'].iloc[-501:]
cierres.pct_change().add(1).cumprod().mul(100).plot(logy=True)
```



Como ven, ambos gráficos, el simulado por montecarlo con un GBM, y el real, no se parecen ni ahí.
Los pego acá los dos juntos para apreciarlo mejor:



Ahora ¿Por qué no se parecen?

En principio partimos de un GBM asumiendo una dist normal que claramente no tienen los retornos, y en segundo lugar porque el efecto de correlación y subtendencias no se refleja en el GBM y si se refleja en el real. Por eso vemos que en marzo-2020 en el momento covid la tendencia bajista super marcada en los 5 activos se ve muy claro en el real y ni de casualidad un GBM va a marcar tendencias tan claras en varios activos al mismo tiempo, si hiciéramos esto para 100 activos el efecto sería mas notorio aunque el gráfico no se entendería tan claro.

Conclusión Tipos de bots vs Backtesting

Bueno, dirán ¿para qué tanta sarasa? ¡Vamos al código, man! Tranquiloss... ya llegaremos. Me pareció importante entender de antemano que tan diferentes son los objetivos de cada tipo de bot, e incluso dentro de una misma familia de bots, diferentes variantes. Con esto me imagino que se darán una idea que no hay un "modelo" o "framework" de backtesting único que aplique a todos los tipos de operatoria.

En realidad hay cosas básicas que no pueden faltar, pero son como una especie de "condición necesaria pero no suficiente" para empezar a hablar de backtesting, y lo que quiero de ahora en adelante es empezar a concentrarnos en ese "cosas necesarias aunque no suficientes" después les iré dando tips y sugerencias para que cada uno vaya armando su propio camino en este mundo de los backtestings, pero al menos veremos esa base a la que de ahora en más voy a llamar "framework" de trabajo, este framework va a tener por un lado "objetivos" y por otro "herramientas" para conseguirlos.

Quiero aclarar que este "framework" de backtesting que les voy a presentar, es un esquema propio, obviamente que fui tomando ideas de diferentes autores, pero quise darle mi impronta y armar algo propio.

Etapas de un backtesting

Si bien no hay ninguna convención al respecto, me animé a armar un esquema de trabajo propio para poder referirme a cada etapa de un backtest ya que, como verán realizar un backtest no es una tarea rápida, simple o estándar, sino que conlleva infinidad de temas a tener en cuenta.

Aclarando que esta clasificación es totalmente arbitraria y personal, les dejo lo que yo considero las etapas básicas de todo backtest y voy a ir haciendo un resumen de cada una antes de meternos de lleno en el código y ejemplos concretos, los ítems nombrados son a modo de exemplificar cada etapa no son muy exhaustivos ni completos:

Etapa de Datos

La primera etapa naturalmente nos vamos a concentrar en nuestra "materia prima": **los datos**

1- Preparación de Datos:

- Selección de fuente/s de datos
- Captura de datos.
- Filtrado, procesado, normalizado de datos
- Definición de sistema de actualización de datos

Etapa de Research / PreBacktest:

Esta etapa es la de prueba básica, aquí vamos a verificar si hay correlación entre nuestros indicadores y lo que hace el precio, y si muy a grandes rasgos me pueden dar alguna mínima ventaja estadística

2- PreBackTest

- Planteo de "Racional" o Hipótesis de trabajo, definir "backtesteabilidad".
- Lógica operacional, parametrización y gatillos
- Data Mining
- Construcción de indicadores/features
- Filtro de datos: Filtros, limpieza de datos, Data Profiling, Survivorship Bias
- Conociendo mejor a los datos: Correlación, autoregresión, homocedasticidad/heterocedasticidad, FIV
- Tabla de posibles trades pasados según racional de trading
- Tabla de resultados o reporting básico
- % de trades positivos y negativos / Esperanza matemática del método
- Contraste de temporalidades, costos transaccionales, etc.
- Tiempo in/out, costo de oportunidad

Etapa de Análisis y métricas

En esta etapa vamos a calcular todo tipo de ratios de la estrategia, que van a ser nuestras "métricas" o KPIs si lo quieren ver como un modelo de negocio, la idea de las "métricas" es tener indicadores comparables de performance para después eventualmente comparar estrategias y decidirse por la mejor a implementar de toda la lista que tengamos

3- Backtest básico con enfoque matricial:

- Trades en un grupo de activos, en un rango de parámetros
- Tabla de resultados x trade
- HeatMaps estacionales. Resultados año a año, mes a mes etc
- Comparación con el buy&Hold / Benchmark
- Análisis de colas y drawdowns
- Ratios de riesgo (Sharpe, Sortino, Calmar, VaR, etc)
- Ratios profit/loss (CAGR, payoff ratio, profit ratio, etc)

Etapa de Parametrización

Esta es la etapa peligrosa donde debemos evitar el overfitting, básicamente acá vamos a analizar la sensibilidad de nuestro método o racional evaluado en alguna métrica que consideremos relevante, en función de los parámetros variables del racional, ya se va a entender mas claro con ejemplos.

4- Análisis de sensibilidad

- Parametrización de variables con Montecarlo
- Cambio de indicadores. Modelos de randomización
- Uso de grupos de control

Etapa de contrastes y validaciones

Esta evidentemente es la etapa más compleja, acá seguramente aparezca mucho más Montecarlo, algoritmos de inteligencia artificial, y pruebas en ambientes simulados pero muy parecidos a los reales, o incluso impactando realmente en el mercado

Dada la infinita variedad de métodos y de tipos de bots, este podría ser un capítulo enorme, pero tomando algunas generalidades que cruzan a la mayoría de tipos de bots, podríamos igual armar un esqueleto de dos grandes grupos de contrastes y validaciones:

- El análisis de portabilidad: Básicamente la idea es
"si esto funciona para A ¿por qué no debería funcionar para B?"
Es decir, supongamos que validamos con buenas métricas un racional para operar AAPL, entonces lo probamos con TSLA, o con todos los del sp500, o lo probamos con las techs o por diversos sectores o tipos de activos o mercados o timeframes y vemos si esto fortaleza la idea o la debilita.
- El análisis en entornos más realistas: Básicamente la idea es
"Ok la teoría o modelo muy lindo, pero que pasa si justo cuando va a comprar pasa tal y cual cosa.."
O sea, los modelos de backtest son simplificaciones muy burdas de la realidad, pero nos permiten comparar, parametrizar y demás, el desafío luego es someter a la idea a entornos más complejos donde puedan contemplarse más cosas

5- Análisis de portabilidad

- KDEs de métricas, usando activos diferentes con el mismo backtest
- Búsqueda de clustering x asset y validación cruzada
- Matrices de correlación cross mkt
- Matrices de correlación cross time-frame
- Comportamiento del racional por régimen (volatilidad, épocas, ciclos etc)

6- Backtest Avanzado. Cambio a Enfoque Event-Driven / Montecarlos / Tests reales

- Contextualización en cartera, risk management y momento del mercado
- Manejo de posición/riesgo, exposición óptima (Kelly, Montecarlo etc)
- Factibilidad técnica (volúmenes, liquidez, spreads, fallas, tiempos etc)
- Testeo en mercado real sin afectar microestructura del mercado
- Testeo afectando microestructura del mercado

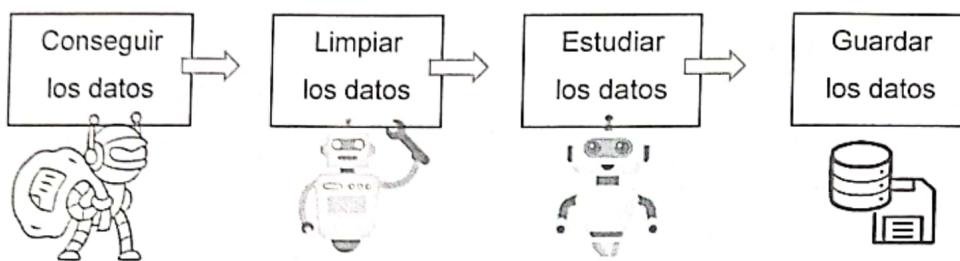
Bien, como les avisé es solo un framework que se me ocurre bastante generalista (es decir que no siempre va a servir tal cual) pero que en muchos casos nos permitirá ordenar bastante los pasos necesarios (pero como dije de entrada no suficientes) a seguir. Analicemos entonces una por una todas estas etapas

Etapa de datos

Esta es la parte más crítica, es la materia prima del backtest, si bien es el inicio de todo, como es un tema muy amplio y que conlleva el uso de muchas herramientas, voy a dejar esta parte para el siguiente tomo, en donde veremos screeners, alertas y demás temas asociados a big-data para trading algorítmico y backtesting, les dejo acá solo un esqueleto de items.

El proceso en orden a seguir podría ser:

- Data Feed: Recolectar los datos.
- Data Profiling: Procesamiento, limpieza, normalización y preparación de datos
- Data Mining: O ingeniería de datos, o sea, hacerlos hablar un poco
- Data Integration: Estructurado, Persistencia y Actualización en bases de datos relacionales



Pero si bien veremos todo este proceso en forma mas ordenada en el siguiente tomo, no quiero dejar de mencionar que en este paso vamos a tener que definir lo siguiente:

- Las fuentes de datos: Nosotros en este tomo para simplificar y que esto sea accesible para todo el mundo usaremos solo APIs gratuitas y públicas como yfinance, y usaré exchanges de cryptos también dada la abundancia y gratuidad de esos datos.
- La forma de extracción de datos: Hay muchas formas desde webscraping, APIs REST, APIs FIX, websockets, servicios internos etc. Como dije la idea de este tomo no es concentrarse en el tema datos, sino en backtesting así que para simplificar usaremos solo APIs REST.
- El análisis y métodos de preprocesamiento de los datos: Aquí es donde filtraremos por ejemplo sesgos de supervivencia, por citar alguno, es decir en lugar de agarrar todo el SP500, vamos a filtrar también categorías como "empresa deslistada del sp500". También aquí normalizamos los datos, los limpiamos, los filtramos etc
- La Persistencia de los datos: Tecnologías, no solo para recopilar, sino para guardar, y para mantener actualizados día a día nuestros datasets.

Fuentes de datos

Las fuentes de datos son un tema complejo porque es el primer punto a analizar y de ello se desprende todo el resto, y como se imaginarán hay fuentes de datos gratuitas y fuentes de datos pagas con un espectro de precios realmente desquiciado, es decir, hay fuentes de datos que cuestan miles de dólares mensuales como Refinitiv, BBG etc.

Obviamente de una fuente de datos gratuitas, o una de pocos usd mensuales a otra de usd 10.000 mensuales, estamos hablando de diferentes calidades de los datos, aunque debo decir que he trabajado con todo tipo de fuentes de datos, y he encontrado grandes baches y errores en información muy cara, así como he encontrado fuentes de datos gratuitas o muy económicas realmente impecables.

Es un tema que da para mucho y lo vamos a profundizar en el siguiente tomo de esta colección.
Les dejo acá simplemente un par de ejemplos para referencia de diferentes fuentes de datos en cada escala de precios porque ya lo tenía de mis clases, es a modo referencial

- Fuentes Gratuitas 100%
 - Páginas o servicios específicos (Ej: Investing, Finviz, StockRow, BarChart, MacroTrends)
 - Google (via google sheets + python). Confiable pero poco práctica
 - Librería YahooFinance. Muy práctica pero Poco confiable
 - API AlphaVantage. Muy limitada 5 requests por minuto
 - APIs Cryptomonedas. En crypto todo es gratis práctico y de excelente calidad
- APIs Freemium rangos de USD 10 a USD 500.
 - IEX cloud (Muy completa y recomendable, pero escala rápido el precio)
 - FMP (Fundamentals, muy confiable y bastante accesible)
 - Finnhub/Tiingo (Buena relación precio/prestaciones)
- APIs Profesionales / Institucionales
 - Gratis para clientes
 - Ameritrade (stocks y opciones USA)
 - IBKR (stocks, bonos y opciones USA)
 - IOL (stocks, opciones y bonos ARG, Byma)
 - Remarks y Primary (Futuros ARG, Rofex)
 - Profesionales e Institucionales, a partir de USD 10.000 mensuales
 - Reuters / Refinitiv
 - BBG

Hay veces que por presupuesto nos tenemos que ajustar a fuentes gratuitas o muy económicas y por lo tanto debemos combinar datos de diferentes fuentes para lograr el dataset que necesitamos, si el presupuesto es ilimitado claramente las fuentes de datos mas profesionales tienen resuelto casi todo en una única integración que nos simplificará mucho el trabajo.

Extracción de datos

Una vez que ya sabemos de donde sacar los datos tenemos que definir con que tecnología vamos a extraerlos, algunos ejemplos:

- One-time web-scraping
- Programmable web-scraping system
- One-time API query
- Programmable API data source
- Online + Offline systems
- Concurrent multi-endpoint data-system

Pero veamos más en detalle de que se trata cada una y algún ejemplo típico

- **One-time web-scraping**

En este caso vamos a barrer de alguna web una info por única vez

Supongamos que quiero ver la correlación de una acción concreta contra algún commodity y necesito alguna serie larga de precios de ese commodity para analizar estacionalidad y correlación contra el activo en cuestión, como es algo que no hacemos periódicamente, lo más probable es que no nos querremos suscribir a alguna fuente de datos paga del commodity ya que es para un uso esporádico, y probablemente encontraremos una buena data en alguna web que no tenga API, por lo que simplemente realizaremos un web scraping una única vez y listo

- **Programmable web-scraping system**

Ahora, también nos puede pasar que necesitamos cierta data que está bastante útil para nuestro uso, pero no tiene API, y la necesitamos con cierta frecuencia de actualización, por ejemplo todas las semanas, personalmente he hecho bastantes sistemas de este tipo para datos de fundamentales, ya que son datos difíciles de conseguir de buena calidad mas allá de los balances, y al tener un screener de mas de 5.000 activos cada semana necesitaba actualizar si o si la base de datos

En estos casos lo que se automatiza es todo, es decir no solo el barrido de datos de la fuente original, sino toda la limpieza, depuración, filtros, normalización etc.

- **One-time API query**

Esto es poco común ya que cuando se necesita una sola fuente de datos una vez, generalmente se recurre a algún servicio donde estén ya disponibles las bases de datos listas para descargar completas, las APIs son mas útiles cuando hay que automatizar la descarga periódica y actualizada de una serie de datos, pero aun así hay veces que es mas confiable o barata la fuente de datos de una API para una descarga por única vez y nos tomamos el trabajo de leer la doc de la API para esta descarga única

- **Programmable API data source**

Esto es lo más común y recomendable, es "más mantenible", ya que el código único de cada query a la API y la conexión a la base de datos está todo en el mismo lugar o script y ya, y a su vez tal vez una API requiere menos mantenimiento que un scraping que depende de las páginas de donde se extraen los datos. Es lo más recomendable para cuando necesitamos periódicamente una serie de datos actualizada y persistirla en una base de datos (ya veremos estos procesos automatizados en el tomo que viene)

- **Online + Offline systems**

Otro sistema muy común pero poco amigable de dataFeed, es un híbrido entre un sistema online + un sistema offline. Llamo offline a un sistema que tiene una cierta data fija en un servidor interno in-company que no depende de consultar datos externos ya sea vía API o vía scraping.

Es un sistema más caro ya que hay que tener una infraestructura interna de servidores offline que es mucho más caro que cualquier sistema cloud.

¿Y Para qué demonios usaría un sistema offline?

Bueno, generalmente el disparador de esta necesidad es la SEGURIDAD, claramente si tengo un sistema que opera mucho volumen y un mínimo hackeo temporal podría llegar a ocasionarme una pérdida enorme, quizás la opción 100% cloud no sea muy recomendable, y lo mejor en ese caso sería un mix donde se minimice o anule el potencial daño con una parte totalmente offline fuera de riesgo de hack o con un riesgo infinitamente menor. Otra razón podría ser data muy costosa que no implica ningún riesgo operativo pero que al ser una BBDD que costó mucho esfuerzo y/o dinero armar, no queremos exponernos a que la competencia acceda a ella.

- **Concurrent multi-endpoint data-system**

Ya para cerrar el tema, hay sistemas de dataFeed más profesionales, generalmente los que tienen las grandes compañías, no porque sea mejor, sino por una cuestión de escala. Estos sistemas tienen arquitecturas de las más variadas, con muchos servidores, con muchos dominios, con muchas bases de datos, etc, hay libros enteros de arquitecturas de infraestructura de datos, aquí solo las menciono para que al menos quede la idea de que podríamos tener por ejemplo:

- Un servidor almacenando una BBDD del mercado equity USA
- Otro servidor almacenando una BBDD del mercado Europeo en general
- Otro servidor que se añade luego que actualiza data de Opciones de USA
- Un cuarto servidor que accede a servicios externos de los anteriores y nuclea toda la data

Si bien el ejemplo es super básico, es muy común que pase que se arma una base de datos y un sistema que lo mantenga actualizado, y luego de un tiempo aparece la necesidad de ampliarlo a más activos o más mercados, y de repente no es tan sencillo o seguro hacerlo en el mismo servidor, o sistema por lo que se hace en un sistema externo, o se subcontrata, y así empiezan a crecer los servidores o servicios de datos externos contratados y en algún momento se arma un nodo concurrente que, digamos en forma simple, "nuclea" toda la data y servicios en un mismo nodo o punto de consulta. Nada, eso, lo dejé mencionado al pasar al menos, porque sin dudas, en algún momento de su vida se van a topar con un sistema así.

Etapa de Research / PreBacktest

Bien, una vez que contamos ya con los datos almacenados, limpios, prolijos, normalizados, revisados, regularizados y debidamente estructurados en una base de datos, es decir una vez que contamos con nuestra materia prima, viene el proceso de empezar a trabajarla, es decir, empezar a tirar las primeras ideas en bruto e ir viendo la forma que toma por decirlo rápido.

Como les adelanté en el punteado de ideas de esta etapa acá hay varias cosas a ir haciendo, pero lo principal, el primer paso sería esta seguidilla de acciones básicas antes de comenzar:

- Definir el racional de la idea de bot
- Definir si es backtesteable o no
- Definir la lógica operacional ("la idea de trading" a llevar adelante)
- Definir la parametrización, y gatillos

Pongamos algunos ejemplos:

Ejemplo 1: Bot de Market Making

Racional del Bot

- Mercado: Crypto en BTC, ETH y ADA
- Exchange ruteo: FTX
- Exchanges referencias: Binance, Coinbase, Huobi
- Costo transaccional: 0%, Rebate o reward medio del orden del 0.01% del volumen operado
- Spread medio instrumentos: 0.15%
- Spread mínimo, máximo, quantiles, distr: XX (análisis paramétrico de la dist de spreads)
- Volumen medio/Día: 10 BTC x hora
- Estimación de agresividad Taker / (Maker+Taker): 15%

Es Backtesteable:

NO, solo es simulable pero, como vimos, no podremos hacer un backtest de este tipo de bot

Lógica operacional:

Acá viene la descripción de las iteraciones y lógicas que el bot llevaría a cabo, en este tipo de bot con lógicas muy iterativas la mejor idea es un diagrama de flujo, la gracia del diagrama de flujo es que nos permite visualizar que decisiones va tomando el bot paso a paso, pero lo veremos esto en el tomo de BOTs, acá nos concentraremos en el backtesting por ahora.

Parametrización y Gatillos:

Esto surge del paso anterior, por ejemplo, a modo de ejemplo, supongamos que en la lógica operacional decía en una parte que "pasado XX% spread el bot debía meterse a cerrar las puntas poniendo más cantidad Qbid/QAsk y mas apertura YY% en la punta más cargada en ZZ% profundidad", bueno ahí los parámetros, XX, YY, ZZ y el ratio QBid/QAsk son valores o gatillos a parametrizar

La gracia de definir bien los parámetros del bot es que son las variables que vamos a buscar modificar para estudiar la sensibilidad del método en la etapa 4 del backtest.

Bueno, la descripción de estos pasos la hago para que se den una idea de a que iba la cosa claramente dado que ya dijimos que el tipo de bot no era backtesteable, no hacía falta seguir.

Ejemplo 2: Bot de swing trading tipo tren following

Racional del Bot

- Mercado: Equity US
- Exchange ruteo: IBKR
- Activos a tradear: Techs de alta liquidez, según screener de volatilidad
- Direccionalidad: Long
- Temporalidad: Diaria
- Gatillo compra: Cruce de medias + Oscilador en zona óptima
- StopLoss: Fijo porcentual desde precio inicial
- Take Profit: No
- Gatillo Salida: Señal cruce negativo

Es Backtestable:

SI

Lógica Operacional:

El Bot rastrea día a día los indicadores (cruce de medias exponenciales AA/BB y oscilador RSIcc > XX) y de darse la condición de entrada, compra el activo al precio de cierre del día que marcó el indicador, La venta se da por stoploss de un YY% fijado de antemano, o bien porque el cruce de medias AA/BB da negativo.

Parametrización

Encontrar buenos valores de AA, BB, CC, XX, YY citados en la lógica operacional

Nota: Si bien el ejemplo es extremadamente burdo, me parece que como primera idea de bot para backtestarlo es interesante para no irme por las ramas y llevar un orden en este framework de backtest que me propuse desarrollar a lo largo del libro, por lo que verán, que varias veces me refiero a este "ejemplito" o similares, mas que nada para dejar el código de algo replicable fácilmente y concentrarnos en el código del backtest en sí. Luego ya habrá tiempo para concentrarnos en las estrategias y en elaborar ideas más potables y usar indicadores más amplios que estos, de en unas páginas más, les dejo una mini-guía de uso de indicadores en trading algorítmico que armé recopilando bibliografía bien diversa.

Bien ahora si una vez definido esto entraríamos de lleno a esta etapa de tanteo de ideas, con los principales puntos de un pre-backtest, que como los enumeramos antes podrían ser:

- Data Mining
- Construcción de indicadores
- Filtro de datos: Filtros, limpieza de datos. Data Profiling. Survivorship Bias
- Introspección de features: Correlación, autoregresión, homocedasticidad/heterocedasticidad, FIV
- Planteo de "Racional" o Hipótesis de trabajo.
- Tabla de posibles trades pasados según racional de trading
- Tabla de resultados o reporting básico
- % de trades positivos y negativos / Esperanza matemática del método
- Contraste de temporalidades, costos transaccionales, etc.
- Tiempo in/out, costo de oportunidad

Data Mining

En esta etapa nos concentraremos en "buscar patrones" o indicadores que sirvan para darnos señales o algo similar, es una etapa larga que requiere mucha práctica y experiencia, en este tomo la voy a resumir poniendo arbitrariamente dos indicadores triviales simplemente a modo de ejemplo, pero la idea es que quien quiera desarrollar un buen bot de trading algorítmico debe investigar dentro del tipo de bot que vaya a realizar muchos indicadores, y a medida que los calcula, ir analizando las correlaciones contra el precio o el retorno futuro del mismo. Mas adelante, de todos modos, les voy a dejar una mini guía de construcción de indicadores y algunas sugerencias al respecto.

Muy a modo de ejemplo, vamos a tomar el ejemplo 2 de la página anterior, donde la idea era backtestear un bot de swing trading del tipo tren-following, con un cruce de medias y el oscilador RSI, yo solo les voy a mostrar lo básico pero acá podrían ampliar el estudio con mas herramientas estadísticas como las vistas en el t[6] o con herramientas de machine learning para búsqueda de patrones, que veremos mas adelante en el tomo de IA. Ahora que lo pienso este tema de búsqueda de patrones da para un tomo propio de data mining, así que lo voy a tener presente.

Bueno, como dijimos, vamos a suponer que lo único que conocemos para buscar patrones son los cruces de medias simples y el RSI, le vamos a agregar la volatilidad histórica y con ello procederemos

Obviamente empezamos bajando la data y seteando los parámetros

```
import yfinance as yf

# Parámetros iniciales
ruedas = 14
ticker = 'AAPL'
fast, slow = 20, 60
sigma = 40

# Descarga data
data = yf.download(ticker, auto_adjust=True, start='2000-01-01')
data
```

Date	Open	High	Low	Close	Volume
2000-01-03	0.803995	0.862449	0.779559	0.858137	535796800
2000-01-04	0.829868	0.848075	0.775726	0.785788	512377600
2000-01-05	0.795370	0.847596	0.789620	0.797286	778321600
2000-01-06	0.813578	0.820285	0.728291	0.728291	767972800
2000-01-07	0.739790	0.774288	0.732124	0.762789	460734400
...
2021-08-24	149.449997	150.860001	149.149994	149.619995	48606400
2021-08-25	149.809998	150.320007	147.800003	148.360001	58991300
2021-08-26	148.350006	149.119995	147.509995	147.539993	48597200
2021-08-27	147.479996	148.750000	146.830002	148.600006	55721500
2021-08-30	149.000000	153.490005	148.610001	153.119995	90956700

5450 rows × 5 columns

Creo que ya lo dije, pero lo repito, el próximo tomo va a estar dedicado exclusivamente a eso, pero acá nos vamos a concentrar solo en lo que respecta a backtesting, con lo cual vamos a traer data de yfinance o algún Exchange crypto, así como viene y trabajaremos con eso como si estuviese perfecto y demás, pero a fin de hacer foco en el tema del tomo presente, obviamente nadie en su sano juicio haría una backtesting serio con estos datos.

Luego, calculamos los indicadores que dijimos que usaríamos:

```
# Cálculo cruce
data['SMA_{fast}'] = data.Close.rolling(fast).mean()
data['SMA_{slow}'] = data.Close.rolling(slow).mean()
data['Cruce_{fast}_{slow}'] = data['SMA_{fast}'] / data['SMA_{slow}'] * 100 - 100

# Cálculo RSI
data['dif'] = data['Close'].diff()
data['win'] = np.where(data['dif'] > 0, data['dif'], 0)
data['loss'] = np.where(data['dif'] < 0, abs(data['dif']), 0)
data['ema_win'] = data.win.ewm(alpha=1/ruedas).mean()
data['ema_loss'] = data.loss.ewm(alpha=1/ruedas).mean()
data['rs'] = data.ema_win / data.ema_loss
data['rsi'] = 100 - (100 / (1+data.rs))

# Cálculo Volatilidad
data['variacion'] = data.Close.pct_change() * 100
data['sigma_{sigma}'] = data['variacion'].rolling(sigma).std() * (sigma**0.5)

# Seleccionamos las columnas que necesitamos y limpiamos
cols = ['Close', 'Cruce_{fast}_{slow}', 'rsi', 'sigma_{sigma}', 'variacion' ]
data = data.loc[:,cols].dropna().round(4)
data
```

Date	Close	Cruce_20_50	rsi	sigma_40	variacion
2000-03-28	1.0666	11.9010	62.9060	26.6871	-0.3134
2000-03-29	1.0421	11.7560	59.1830	26.5341	-2.2911
2000-03-30	0.9640	11.5426	49.1072	27.7336	-7.4943
2000-03-31	1.0412	11.3689	56.9219	28.4310	8.0020
2000-04-03	1.0220	11.0852	54.6896	28.2821	-1.8409
...
2021-08-26	147.5400	4.7154	52.9573	7.7144	-0.5527
2021-08-27	148.6000	4.5083	55.9559	7.7320	0.7185
2021-08-30	153.1200	4.4472	65.9288	8.0523	3.0417
2021-08-31	151.8300	4.2924	61.6391	8.0207	-0.8425
2021-09-01	152.5100	4.1787	63.0056	7.8535	0.4479

5393 rows × 5 columns

Es importante que ya desde el principio, es decir desde estos primeros pasos, empiezemos a trabajar prolijos, en realidad habría que ir metiendo todo este código dentro de funciones, pero arranquemos así al menos. Lo importante es dejar los parámetros afuera y que no nos quede todo el script hardcodeado, es decir con los números de parámetros ahí sueltos adentro del código que calcula cosas

¿Por qué?

Muy sencillo, porque luego la idea va a ser estudiar la sensibilidad de estos parámetros, con lo cual tendremos que pasar estos parámetros (por ejemplo, ruedas=14 para la ventana del RSI) a funciones que lo vayan cambiando

Es interesante que vayan siguiendo el código y lo vayan replicando, la verdad parece medio mala onda, pero prefiero no ponerles el código en un colab o github, para que se sienten a tiparlo, porque si hacen un copy/paste, no sirve igual, la mejor forma de ir digiriendo el código es ir paso a paso tipeando línea a línea y entendiendo cada paso por separado

Look-Ahead Bias

Bueno, ahora que ya tenemos los indicadores y una variación viene una pregunta crucial, tomemos por ejemplo la línea del 31 de marzo de 2000 que marqué, miren ese RSI de 56.92 y esa variación del +8% ese día:

Si yo operaba ese día porque esperaba, supongamos, un RSI que pase de menos de 55 a mas de 55, y ese día aparecía esa señal... ¿El rendimiento de 8% va adentro de mi trade?

Claramente la respuesta es NO, ya que ese valor del RSI solo lo sé al cierre de esa rueda, por lo tanto tomar en un backtest, o en una prueba de pre-backtest ese rendimiento ese día es un ERROR muy pero muy común y se lo conoce como "Look-Ahead Bias"

Que es ni mas ni menos que suponer que sabemos algo que en realidad no podríamos saber. Es decir "usar datos del futuro"

Ahora ustedes dirán ¿Cómo arreglo esto?

Bueno, a ver, el día ese 31-3-2000 yo sabía el RSI del cierre anterior, eso sí, así que si quiero tomar el rendimiento del día 31-3 tendría que tomarlo con los indicadores del 30-3, es decir adelantar una fila los indicadores, o bien atrasar una fila los rendimientos (por experiencia es más simple adelantar las columnas de los indicadores, es decir, "operar hoy con los indicadores de ayer" para poder tomar el rendimiento de hoy dentro de mi trade)

Como arreglamos esto entonces:

```
data.iloc[:,1:-1] = data.iloc[:,1:-1].shift()  
data
```

Date	Close	Cruce_20_60	rsi	sigma_40	variacion
2000-03-28	1.0688	NaN	NaN	NaN	-0.3134
2000-03-29	1.0421	11.9010	62.9060	28.8871	-2.2911
2000-03-30	0.9840	11.7580	59.1830	26.5341	-7.4943
2000-03-31	1.0412	11.5428	49.1672	27.7386	8.0020
2000-04-03	1.0220	11.3689	56.9219	28.4310	-1.8409
...
2021-08-26	147.5400	4.9216	55.6803	7.6799	-0.5527
2021-08-27	148.6000	4.7154	52.9573	7.7144	0.7185
2021-08-30	153.1200	4.5083	55.9559	7.7320	3.0417
2021-08-31	151.8300	4.4472	65.9288	8.0523	-0.8425
2021-09-01	152.5100	4.2924	61.6391	8.0207	0.4479

5393 rows × 5 columns

Fijense que ahora el 31-3 si yo quiero tomar el rendimiento de ese día, es decir la variación desde el cierre anterior (30-3) al cierre actual del 31-3, en realidad para tomar todo ese movimiento, yo tendría que haber decidido el cierre del 30-3, por eso ahora me queda bien emparejada la matriz, con el indicador de RSI aun en 49.1672 el 31 de marzo porque es el valor que yo tenía el 30-3 (el momento de la decisión)

Si no comprendieron del todo este tema, les pido por favor lo repasen, parece medio un trabalengua, pero posta es super importante, es el típico BIAS que es fácil de cometerse al codear y después el backtest nos da que somos Gaidel y es porque estamos haciendo trampa sin querer.

Bien, ahora que tenemos ciertos "features" o indicadores

- Cruce_20_60
- RSI
- Sigma_40

Y tenemos una **variación**, podríamos inferir si los features son indicativos de algo o podrían servirme de señal ¿o no? mmm, bueno, en realidad hay algo, pero vamos paso a paso, luego retomo esta pregunta, pero veamos en principio como puedo inferir con los features y la variación si estos indicadores me pueden servir de señal o no

La tarea a realizar es ver si hay **correlación** o no, y si me generan algún tipo de **clasificación diferenciable**, vamos a cada punto

Correlación features vs variación

Bien, una de los puntos a investigar en este pre-backtest es la correlación de los indicadores con los retornos del activo, ok, veamos cómo nos da:

Definimos una lista de features que son los indicadores de nuestro dataframe y calculamos las r^2

```
features = ["Cruce_20_60", "rsi", "sigma_40"]

for feature in features:
    r2 = data[feature].corr(data.variacion)
    print(f"El r2 con {feature} es {r2:.2f}")

El r2 con Cruce_20_60 es 0.02
El r2 con rsi es 0.30
El r2 con sigma_40 es -0.05
```

El cruce de medias no me dice nada, la volatilidad tampoco, pero el RSI me está dando un r^2 de 0.3

Alguno me dirá:

"Juan, el r^2 de 0.3 es muy débil, busco mejor un r^2 de 0.9 o similar"

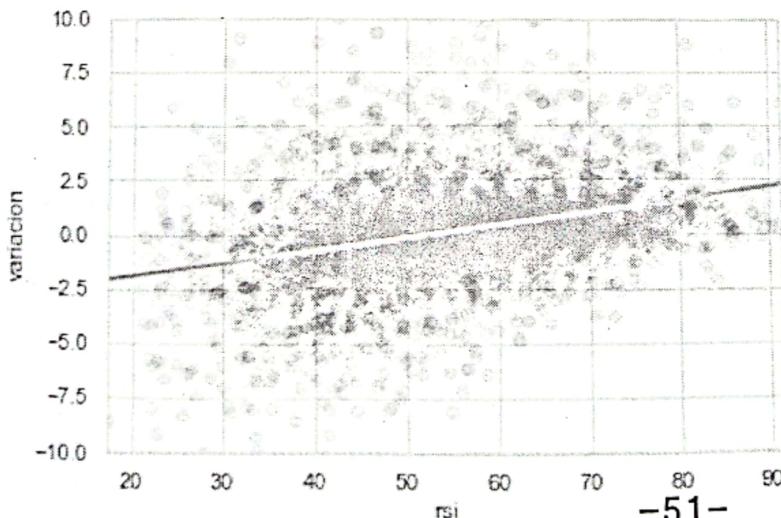
Ok, sí, es muy débil, si estuviera usando esto para correlacionar un síntoma a un diagnóstico en medicina para un sistema automatizado es muy débil y debería descartarse, pero estamos en finanzas, intentando un modelo de trading algorítmico, y como dijimos, los retornos siguen un proceso estocástico con una cantidad de ruido blanco enorme, con lo cual, cualquier indicador que me de una señal con una correlación mayor a 0.1 diría que es al menos para tener en cuenta como posible feature, después vamos mejorando y viendo como combinar todo lo que tengamos para lograr el mejor sistema posible, pero en principio ese 0.3 es algo "potable" o "aprovechable"

Bueno, veamos la regresión entonces como se ve en un gráfico con seaborn (esta herramienta la vimos en el tomo 6, como muchas de las que iré usando en este tomo, así que si se pierden en el código o algún concepto estadístico, revean ese tomo):

```
import seaborn as sns
sns.set(style = 'whitegrid')

g = sns.regplot(x="rsi", y="variacion", data=data, line_kws={"color": "red"}, scatter_kws={"color": "k", "alpha":.5})

g.set(ylim=(-10, 10))
```



¿Qué me dice esto? Por ahora nada, no tengo ningún racional a backtestear ni ningún parámetro para el RSI, pero al menos se que el valor del RSI tiene cierta correlación con la variación diaria, lo ideal sería empezar a darle forma a la idea de trading para poder avanzar

En el gráfico se ve que la correlación es positiva, es decir que cuando el RSI anda alto en promedio ese dia AAPL tiene un mejor retorno que cuando anda bajo, pero eso no alcanza para nada.

Yo necesitaría una lógica por ejemplo comprar con el RSI arriba de XX valor y holdear XX días o holdear hasta que se de tal gatillo etc.. Ahora ahí hay un pequeño detalle, si la señal tarda 5 días en aparecer, por ejemplo, yo estoy a ciegas, es decir, esta correlación positiva RSI/Retorno es para la variación de 1 día, no tengo ni idea que pasa más adelante

¿Cuánto más adelante?

Es relativo, pero si tenemos indicadores como el RSI que parametrizamos en 14 días de ventana, y un cruce de 20 y 60 ruedas, y una volatilidad en 40 ruedas atrás, podemos decir que nuestra ventana forward puede andar entre 10 y 60 días tranquilamente, mas o menos una ventana similar a la que tomamos hacia atrás de datos. Esto no es una regla escrita en ningún lado, pero es razonable, puede ser un poco mas o un poco menos, pero claramente poco me va a servir un indicador que arrastre 15 días de datos, para predecir un movimiento a 1 año, o viceversa

Calculemos entonces dos variaciones a futuro a las que llamaremos "forwards" o "fw_xx" siendo XX la cantidad de días o períodos hacia adelante

Volvemos a descargar los datos entonces:

```
import yfinance as yf
import numpy as np

# Parámetros iniciales
ruedas = 14
ticker = 'AAPL'
fast, slow = 20, 60
sigma = 40

# Descarga data
data = yf.download(ticker, auto_adjust=True, start='2000-01-01', end='2021-09-02')
data
```

Date	Open	High	Low	Close	Volume
2000-01-03	0.803995	0.862449	0.779559	0.858137	535796800
2000-01-04	0.829868	0.848075	0.775726	0.785788	512377600
2000-01-05	0.795370	0.847596	0.789620	0.797286	778321600
2000-01-06	0.813578	0.820285	0.728291	0.728291	767972800
2000-01-07	0.739790	0.774288	0.732124	0.762789	460734400
...
2021-08-26	148.350006	149.119995	147.509995	147.539993	48597200
2021-08-27	147.479996	148.750000	146.830002	148.600006	55721500
2021-08-30	149.000000	153.490005	148.610001	153.119995	90956700
2021-08-31	152.660004	152.800003	151.289993	151.830002	86453100
2021-09-01	152.830002	154.979996	152.339996	152.509995	80313700

5452 rows x 5 columns

Volvemos a calcular los indicadores, pero agregamos esta vez los forwards

```
# Cálculo cruce
data[f'SMA_{fast}'] = data.Close.rolling(fast).mean()
data[f'SMA_{slow}'] = data.Close.rolling(slow).mean()
data[f'Cruce_{fast}_{slow}'] = data[f'SMA_{fast}'] / data[f'SMA_{slow}']*100 -100

# Cálculo RSI
data['dif'] = data['Close'].diff()
data['win'] = np.where(data['dif'] > 0, data['dif'], 0)
data['loss'] = np.where(data['dif'] < 0, abs(data['dif']), 0)
data['ema_win'] = data.win.ewm(alpha=1/ruedas).mean()
data['ema_loss'] = data.loss.ewm(alpha=1/ruedas).mean()
data['rs'] = data.ema_win / data.ema_loss
data['rsi'] = 100 - (100 / (1+data.rs))

# Cálculo Volatilidad
data['variacion'] = data.Close.pct_change() * 100
data[f'sigma_{sigma}'] = data['variacion'].rolling(sigma).std() * (sigma**0.5)

#forwards
data["fw_10"] = (data.Close.shift(-10) / data.Close.shift() -1)*100
data["fw_20"] = (data.Close.shift(-20) / data.Close.shift() -1)*100

# Seleccionamos las columnas que necesitamos y Limpiamos
cols = ['Close',f'Cruce_{fast}_{slow}', 'rsi',
        f'sigma_{sigma}', 'variacion', 'fw_10', 'fw_20' ]

data = data.loc[:,cols].dropna().round(4)
```

Date	Close	Cruce_20_60	rsi	sigma_40	variacion	fw_10	fw_20
2000-03-28	1.0666	11.9010	62.9060	26.6871	-0.3134	-14.4201	-13.0766
2000-03-29	1.0421	11.7560	59.1830	26.5341	-2.2911	-21.4736	-8.8950
2000-03-30	0.9640	11.5426	49.1672	27.7336	-7.4943	-16.2759	-8.7357
2000-03-31	1.0412	11.3689	56.9219	28.4310	8.0020	-11.0338	-1.1432
...
2021-07-30	145.6418	8.4181	58.3286	7.5747	0.1511	2.5291	2.1853
2021-08-02	145.3023	8.4098	57.3249	7.4469	-0.2331	3.7614	5.1347
2021-08-03	147.1395	8.3780	61.2147	7.4902	1.2644	3.3638	4.4925
2021-08-04	146.7302	8.1974	59.9045	7.5154	-0.2782	-0.5298	3.6499

Y volvemos a desplazar las columnas de los "features" [1:-1]

```
data.iloc[:,1:-1] = data.iloc[:,1:-1].shift()
data
```

Date	Close	Cruce_20_60	rsi	sigma_40	variacion	fw_10	fw_20
2000-03-28	1.0666	NaN	NaN	NaN	NaN	NaN	-13.0766
2000-03-29	1.0421	11.9010	62.9060	26.6871	-0.3134	-14.4201	-8.8950
2000-03-30	0.9640	11.7560	59.1830	26.5341	-2.2911	-21.4736	-8.7357
2000-03-31	1.0412	11.5426	49.1672	27.7336	-7.4943	-16.2759	-1.1432
...
2021-07-30	145.6418	8.3399	57.8856	7.7463	0.4552	2.8508	2.1853
2021-08-02	145.3023	8.4181	58.3286	7.5747	0.1511	2.5291	5.1347
2021-08-03	147.1395	8.4098	57.3249	7.4469	-0.2331	3.7614	4.4925
2021-08-04	146.7302	8.3780	61.2147	7.4902	1.2644	3.3638	3.6499

Y ahora que tenemos estas dos nuevas columnas que son los forwards de 10 y 20 días hacia adelante, calculamos la correlación de nuestros mismos 3 features con ellos

Con el fw_10 primero:

```
features = ["Cruce_20_60", "rsi", "sigma_40"]

for feature in features:
    r2 = data[feature].corr(data.fw_10)
    print(f"El r2 con {feature} es {r2:.2f}")

El r2 con Cruce_20_60 es 0.06
El r2 con rsi es 0.14
El r2 con sigma_40 es -0.11
```

Y con el fw_20 luego:

```
features = ["Cruce_20_60", "rsi", "sigma_40"]

for feature in features:
    r2 = data[feature].corr(data.fw_20)
    print(f"El r2 con {feature} es {r2:.2f}")

El r2 con Cruce_20_60 es 0.07
El r2 con rsi es 0.05
El r2 con sigma_40 es -0.12
```

Como ven, me bajó bastante el r^2 contra el RSI ahora mas contra el fw20 que contra el fw10, pero sin embargo contra el sigma_40 ya me empieza a correlacionar un poco más, esta vez en forma inversa, es decir a más volatilidad peor performance.

Otra forma más pythónica, todo de una de ver las correlaciones pudo haber sido:

```
data.corr().iloc[-2:, 1:-3].round(2)
```

	Cruce_20_60	rsi	sigma_40
fw_10	0.06	0.14	-0.11
fw_20	0.07	0.05	-0.12

Es decir, en lugar de ir calculando feature contra fw uno por uno, calculo toda la matriz de covarianzas o en este caso de correlación, y luego con el iloc[] selecciono las filas y columnas que me interesan de toda la matriz

O esta mas expresiva si quieren con el loc[]

```
features = ["Cruce_20_60", "rsi", "sigma_40"]
data.corr().loc[['fw_10','fw_20'], features].round(2)
```

	Cruce_20_60	rsi	sigma_40
fw_10	0.06	0.14	-0.11
fw_20	0.07	0.05	-0.12

Bien, queda claro que van a buscar indicadores con una correlación un poco mejor que 0.10 pero como les decía antes conforman con indicadores con un r^2 de 0.3 o similares que es muy bueno dada la cantidad tan preponderante de ruido blanco en los movimientos.

Respecto a la cantidad de días o velas hacia adelante para correlacionar los features, dependerá de que tan largos o cortos sean los swings que planean hacer con el bot, obviamente para un bot que tenga una media de 1 día entre compra y venta, no hacen falta los fw que vimos, sino que solo con la variación de 1 día alcanza para estimar que tan útil es el feature.

Clasificaciones diferenciables

Visto el tema de las correlaciones, queda otro aspecto a analizar que lo complementa que es la capacidad del feature de clusterizar los datos, haciendo una especie de ingeniería inversa, es decir si tomamos por ejemplo todas las filas cuya variación expost (fw) es positiva por un lado y las que es negativa por el otro, si el indicador refleja alguna diferencia o no. De hacerlo es un buen clasificador (quizá no correlacione mucho, pero para modelos con mucha clasificación sean muy útiles, mas adelante en el tomo de machine learning, veremos modelos de clasificación como los árboles de decisión a los cuales les vienen muy bien este tipo de features)

Vean esto:

```
data['variacion_tipo'] = np.where(data.variacion > 0, 'Up', 'Down')

medias = data.groupby('variacion_tipo').mean().loc[:,features].round(2)
desvios = data.groupby('variacion_tipo').std().loc[:,features].round(2)

print(f'Medias según clasificación de Variación \n {medias} \n')
print(f'Desvíos según clasificación de Variación \n {desvios} \n')

Medias según clasificación de Variación
    Cruce_20_60      rsi      sigma_40
variacion_tipo
Down          1.64 [ 51.14 ]     14.57
Up            2.29 [ 58.78 ]     13.80

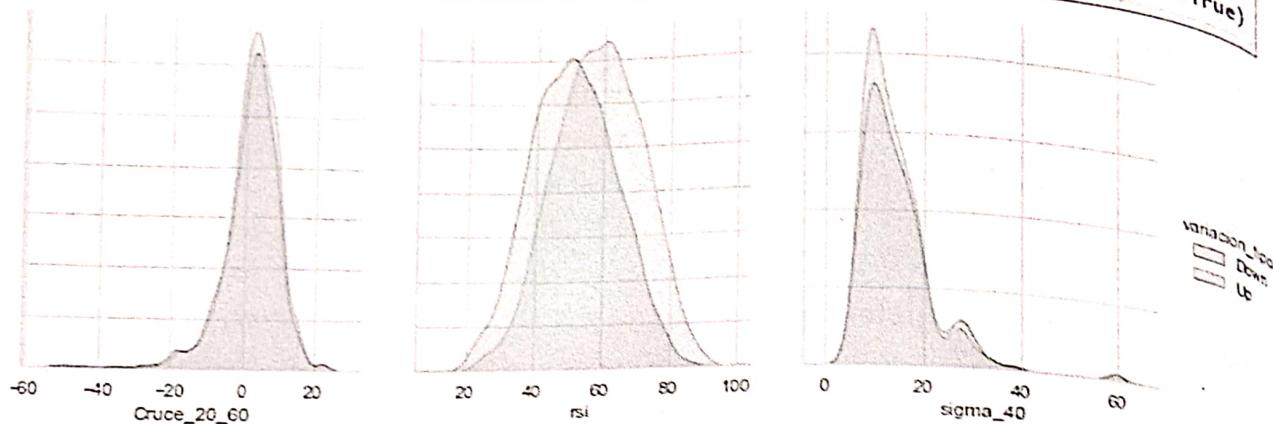
Desvíos según clasificación de Variación
    Cruce_20_60      rsi      sigma_40
variacion_tipo
Down          8.28 [ 12.47 ]     7.70
Up            7.78 [ 12.68 ]     6.87
```

A priori mirando solo las medias, pareciera que el cruce_20_60 es el mejor clasificador de los 3, ya que pasa de 1.64 a 2.29, aumenta un 40% mucho más que por ejemplo el RSI que pasa de 51.14 a 58.78 o sea aumenta un 15%, pero hay que medirlo en términos de cuantos desvíos estándar varían según el clasificador, en este caso el cruce varía menos de 0.1 desvíos, mientras que el RSI varía 0.64 desvíos, es decir que el RSI me permitirá clasificar mucho mejor los que suben de los que bajan

Obviamente lo ideal serían 2 desvíos para que casi ni se solapen los dos grupos, pero eso en la bolsa no existe, pero con que encontramos un clasificador que separe ambas distribuciones mas de medio desvío es algo mas que aceptable como clasificador.

Pero se ve mejor en un gráfico tipo así

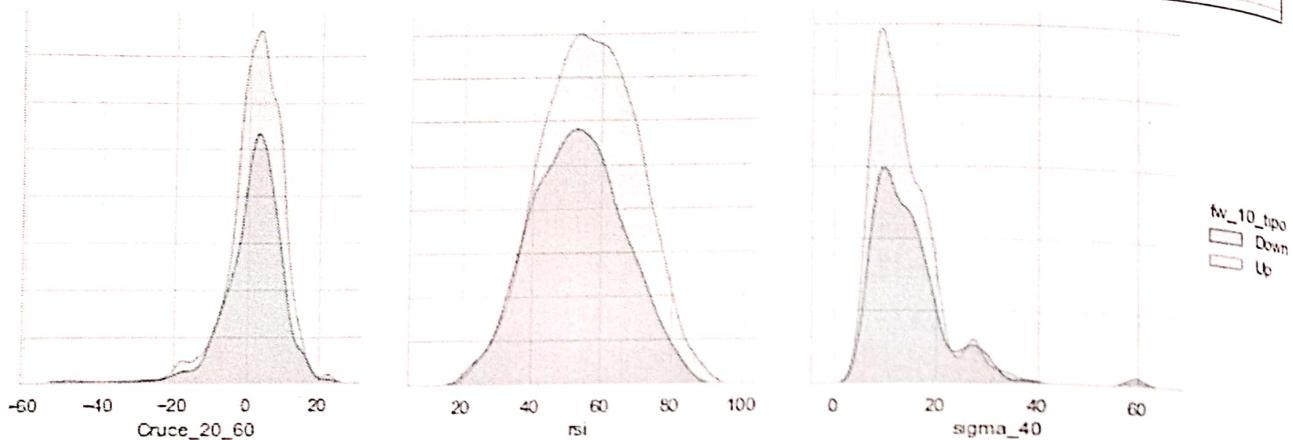
```
for feature in features:  
    sns.displot(data, x=feature, hue="variacion_tipo", kind="kde", height=4, fill=True)
```



Aquí se aprecian las distribuciones de los features clasificadas exposito según si la variación fue negativa o positiva y se nota lo que decíamos que efectivamente el RSI es el mejor clasificador de los tres, ya que tiene un desfasaje de 0.65 desvíos entre un grupo y otro.

Lo mismo contra los fw_10 y 20, les hago el primero el segundo se los dejo a ustedes:

```
data['fw_10_tipo'] = np.where(data.fw_10 > 0, 'Up', 'Down')  
  
for feature in features:  
    sns.displot(data, x=feature, hue="fw_10_tipo", kind="kde", height=4, fill=True)
```



Medias según clasificación de Fw_10
Cruce_20_60 rsi sigma_40

fw_10_tipo	Cruce_20_60	rsi	sigma_40
Down	1.54	53.04	14.96
Up	2.29	56.54	13.63

Desvíos según clasificación de Fw_10
Cruce_20_60 rsi sigma_40

fw_10_tipo	Cruce_20_60	rsi	sigma_40
Down	8.58	12.88	8.17
Up	7.62	13.14	6.57

Construcción de indicadores

Esta parte de la etapa de research es la mas "creativa" ya que es una etapa en la que hay que pensar de donde sacamos algún feature que me pueda servir a la hora de gatillar una acción algorítmica, está claro que en los bots de market-making, los de arbitrajes y ese tipo de bots esta etapa no existe, por el contrario, esta etapa es mas para los bots que buscan un profit basado en algún tipo de gatillo de compra y venta.

Si bien este es un tomo de backtesting, le voy a dedicar un capítulo al tema indicadores, porque son, luego de los datos originales, lo más importante a la hora del backtest, hay que saber muy bien como están construidos y como construir los propios

¿Qué tipo de predictores existen?

Ufff, no se por donde empezar a enumerar todos, así que opté por una especie de clasificación de tipos de indicadores, nuevamente, es un invento propio, pero basado en diversa bibliografía, de cada libro saco algo y armo mi propio esquema, van a ver que en este tipo de temática repito mucho lo mismo. En fin vamos a la clasificación:

En principio armé como categorías de indicadores que serían las siguientes:

- Indicadores sobre serie de precios
 - Indicadores tardíos (trend-following)
 - Indicadores de saturación (contrarians)
 - Indicadores de referencia: Osciladores acotados (sobre precio o sobre indicadores)
- Indicadores de flujos (volumen y derivados)
- Indicadores combinados
- Conteos discretos
- Estacionalidad
- Indicadores Estadísticos
- Referenciales (Benchmarks)
- Ratios y series de Análisis Fundamental
- Sentiment
- Exógenos al mercado

Ese sería mi Top10 de categorías principales, con esta clasificación para ordenarlos podemos enumerar todo tipo de indicadores usados en la industria, pero vamos una por una

Indicadores sobre serie de precios

Bueno, esta es la categoría mas trillada digamos, la que aparece en los gráficos de tradingView y la inmensa mayoría de traders humanos usa en el día a día, no están mal, pero digamos es la punta del iceberg en este tema, lo mas rico viene en las otras 7 categorías sin lugar a dudas, sin embargo los indicadores de esta categoría son simples de comprender, tienen mucha historia y están super

extendidos en el uso por lo que son los que mas voy a tomar de ejemplo porque además están en todo tipo de herramienta como tradingView y demás, mientras que los otros tipos van a ser construcciones más personalizadas y serán mas difíciles de encontrar en ese tipo de herramientas populares.

Como dijimos, a su vez este tipo de indicadores basados en las series de precios los podemos agrupar en 3 categorías, acá en realidad hay muchas más, pero estas 3 son muy pero muy usadas así que arrancamos por ahí

- Indicadores tardíos (trend-following)
- Indicadores de saturación (contrarians)
- Indicadores de referencia: Osciladores acotados

Indicadores tardíos

Como indicadores tardíos están todo tipo de medias móviles y cruces de medias móviles y todo indicador construido a partir de ellas (Caso especial son los osciladores acotados que los veremos más adelante)

Empecemos con las medias móviles

Bueno, no hay mucho para decir ¿seguro?

No, es todo un tema super rico, generalmente se usan la media móvil simple SMA o la exponencial EMA y no mucho más, aunque hay otras 5 variedades, pero lo mas importante es entender las diferencias sobre todo al analizar el tema "timing" de las estrategias y buscar mejorarlo



Empecemos por los tipos de medias móviles:

- SMA: La simple (el promedio de una ventana de "n" valores)
- EMA: Exponencial (fórmula de peso $w=2/i+ 1$)
- WMA: Ponderada (fórmula lineal de peso $w=1-i/n$)
- TRIMA: Media móvil simple de media móvil simple SMA_{SMA}
- DEMA: Combinación de EMA y EMA_{EMA}
- TEMA: Combinación de EMA, EMA_{EMA} y EMA_{EMA} _{EMA}
- KAMA: Adaptativa (usa 3 períodos para una KAMA)

Y hay otros como la mesa adaptativa y un par más, pero ya demasiado rebuscadas a mi gusto, de hecho, la adaptativa de Kauffman me parece muy sucia de hecho por el solo punto de necesitar 3 parámetros para una media móvil que son los 3 períodos que usa, pero se las dejo igual para quien tenga la inquietud aunque solo les muestro las otras 6 en comparación con un caso real.

Las tres primeras son puras, mejor dicho, una combinación lineal solo sobre la serie de precios original. Mientras que las otras 4 son construcciones de medias previamente calculadas.

Veamos entonces las 3 puras en detalle sus diferencias

Claramente son promedios, solo que, con diferente ponderación, mientras que la SMA es una media simple, es decir todos los datos pesan igual, en la EMA o exponencial los últimos pesan más que los más antiguos, pero con decaimiento exponencial, en cambio en la WMA el decaimiento es lineal

Veamos los ponderadores primero

```
import matplotlib.pyplot as plt

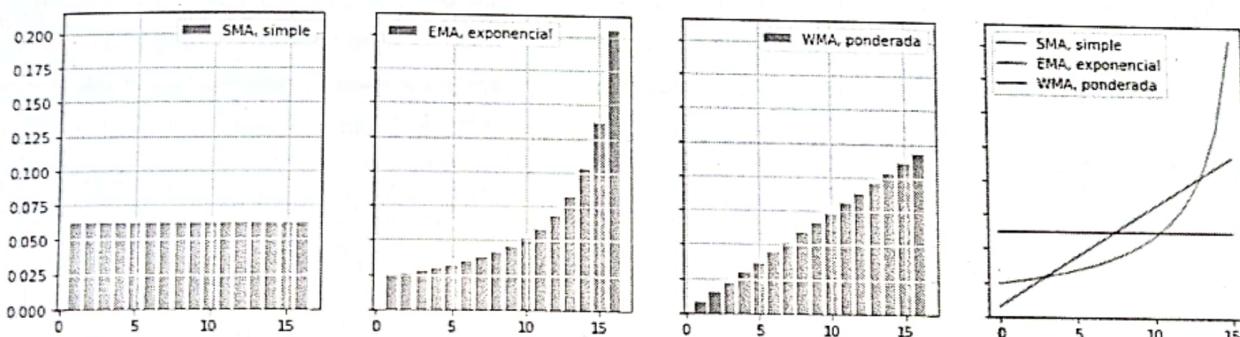
n = 16
w_sma = [1/n]*n
w_wma = [i/sum(range(n+1)) for i in range(1,n+1)]
w_ema = [2/(i+1) for i in range(1,n+1)]
w_ema = [wi/sum(w_ema) for wi in w_ema][::-1]

fig, ax = plt.subplots(figsize=(15,4), ncols=4, sharey=True)

series = {'SMA, simple':w_sma,
          'EMA, exponencial':w_ema,
          'WMA, ponderada':w_wma}

df = pd.DataFrame()
for i, (titulo, serie) in enumerate(series.items()):
    ax[i].bar(range(1,n+1), serie, label=titulo)
    ax[i].legend()
    ax[i].grid()
    df = pd.concat([df,pd.DataFrame(serie)],axis=1)

df.columns=series.keys()
df.plot(ax=ax[3])
```



Se ve claramente como la SMA pesan todos los valores de la muestra en este ejemplo 18, de la misma forma, mientras que en la EMA es exponencial el crecimiento a medida que está más cercano (el último valor, o sea el 18 es el más cercano) y en la WMA se ve que la ponderación crece a medida que está más cerca pero en forma lineal.

Como les decía los otros tipos son construcciones sobre estos básicos, veamos directamente el cálculo en el código, la TRIMA es la media móvil de una media móvil, hay TRIMA lineal y exponencial, y luego la DEMA es: $DEMA[N] = 2 \times EMA[N] - EMA\text{ de }EMA[N]$

El cálculo de la TEMA_[n] es: = $3 \times EMA^{*1} - 3 \times EMA^{*2} + EMA^{*3}$
Siendo:

- $EMA^{*1} = EMA_{[n]}$
- $EMA^{*2} = EMA_{[n]} (EMA^{*1})$
- $EMA^{*3} = EMA_{[n]} (EMA^{*2})$

Veamos entonces el código de todas estas medias móviles

```
import yfinance as yf
import numpy as np
import pandas as pd

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04')
n = 9

data['SMA'] = data.Close.rolling(n).mean()

def wma(serie):
    n = len(serie)
    f = [i/sum(range(n+1)) for i in range(1,n+1)]
    return np.array(serie).dot(f)

data['EMA'] = data.Close.ewm(span=n).mean()
data['WMA'] = data.Close.rolling(n).apply(wma)
data['DEMA'] = data.EMA * 2 - data.EMA.ewm(span=n).mean()
data['TRIMA_L'] = data.SMA.rolling(n).mean()
data['TRIMA_E'] = data.EMA.ewm(span=n).mean()
data['TEMA'] = 3*data.EMA - 3*data.TRIMA_E + data.TRIMA_E.ewm(span=n).mean()

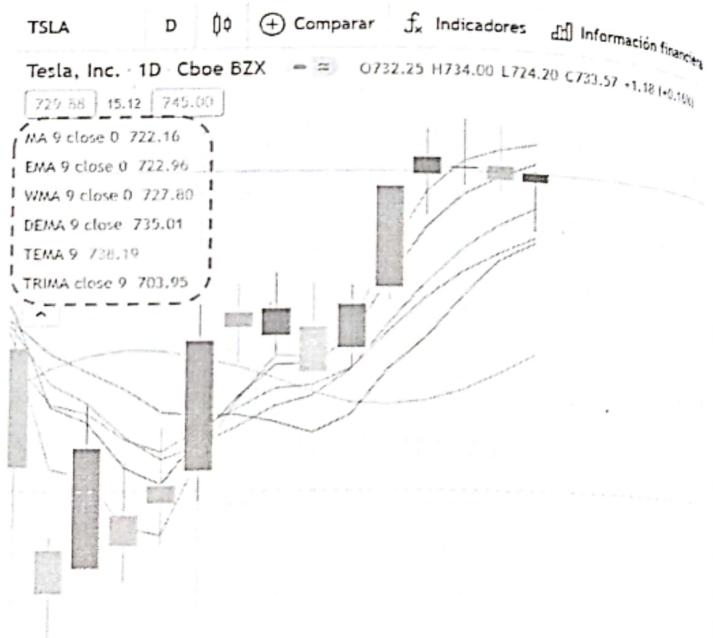
data.drop('Volume', axis=1, inplace=True)
data.dropna(inplace=True)
data.iloc[-1, 4:].round(2)

data
```

SMA	722.16
EMA	722.96
WMA	727.80
DEMA	735.01
TRIMA_L	703.95
TRIMA_E	710.92
TEMA	738.19
Name:	2021-09-03 00:00:00

Les pego esta captura de tradingview para corroborar los valores calculados al 4 de septiembre de 2021 con TSLA

Como ven todas las medias móviles son indicadores "tardíos" es decir que reaccionan tarde a grandes movimientos, algunas un poco mas rápido que otras, pero todas tardías.



Es por ello que son indicadores para estrategias mas bien del tipo tren-following ya que la idea es detectar tendencias cuando se dan "cruces" de medias mas rápidas sobre medias mas lentas, esto quiere decir que se lanza una tendencia, mientras mas cruces se den, mas segura es la tendencia

A partir de estas medias móviles se construye todo tipo de indicadores de momentum y del tipo contrarians, siempre son construcciones sobre medias móviles que se usan de formas diferentes y con diferentes interpretaciones

No me voy a poner a escribir el código de cada uno, ya que no es un libro de indicadores técnicos, pero al menos los clasifico y les dejo un mini renglón de que es cada uno para que investiguen sobre los más usados y voy a escribir y desarrollar solo uno de cada grupo a modo de representar su grupo.

Momentum (tren-following)

- MACD: Operador de EMAs, oscilador
- ADX: Indicador de tendencia, conocido como Average Directional Movement Index
- DX: Es el ADX pero sin suavizar
- ADXR (Promedio del ADX contra el ADX de "n" ruedas atrás)
- MINUS_DI, PLUS_DI, MINUS_DM, PLUS_DM (Componentes del ADX)
- SAR (famoso Parabolic SAR, se usa como Trailing StopLoss para cerrar posiciones)
- MOM (El famoso "Momentum", es la diferencia entre el precio y el precio "n" ruedas atrás)
- MIDPOINT (El promedio entre el Máximo y Mínimo de una serie de "n" velas sobre OHLC)
- MIDPRICE (El promedio entre el Máximo "High" y Mínimo "Low" de la serie de "n" velas)

Osciladores trend-following

- WILLR (Oscilador de Williams, similar al estocástico)
- APO (Absolute price oscillator, Es una diferencia de medias móviles, simples, exponenciales, etc)
- PPO (Percentage price oscillator, Es la diferencia porcentual de las medias móviles)
- STOCH, STOCHF, El conocido oscilador estocástico sobre el OHLC
- RSI: Oscilador de EMAs
- BOP (Balance of power) dado el OHLC es la media móvil de (C-O)/(H-L)
- CMO (Chande momentum oscillator, es una variante del oscilador RSI)
- ROC (Es como el Momentum pero %, mide variación % respecto a "n" ruedas atrás)
- MFI (Money flow index, es un oscilador que se interpreta como el RSI y que tiene en cuenta el volumen)
- TRIX (Es un oscilador de la TEMA respecto de "n" ruedas atrás)
- ULTOSC (Ultimate oscillator, es un oscilador de 3 períodos, como promediar 3 RSIs con 3 "n" diferentes)

Contrarians

- CCI (Commodity channel index, Indica inicio y fin de tendencias, zonas de sobrecompra / sobreventa)
- AROON (Anticipador de inicios de tendencias compuesto por dos curvas, construcción discreta)
- AROONOSC (Es la diferencia entre las curvas del AROON)
- BBANDS (Famosas bandas de Bollinger, bandas de rango de movimiento a +/- 2 sigma de media móvil)
- TRANGE: Proxy de "volatilidad" True range, es un estimador del rango normal de trading o movimiento
- ATR (Average true range, es la media móvil del True Range, es decir la media móvil del rango de movimiento, proxy de la volatilidad, pero en términos absolutos)

Arranquemos con el grupo de los trend followings, y tomaremos el famoso MACD, tomo este a propósito porque me viene bien luego para mostrarles como transformar un indicador de dominio no acotado en un oscilador, pero arranquemos viendo como construir el indicador, y desmenuzaremos un poco la construcción para entenderlo, insisto, tomo este como referencia, pero deberían hacer lo mismo con todos antes de usarlos.

Construcción del indicador

Como vemos, el MACD tiene 3 componentes

- El macd: EMA_rapida [default=12] - EMA_lenta [default=26]
- La Señal: EMA_del_macd [default=9]
- El Histograma: macd – señal

Como pueden ver en la fórmula ya arranca "viciado" de nominalidad, porque no hace ratios o cocientes relativos, sino una resta, con lo cual todo el indicador está construido sobre los valores nominales del subyacente, esto hace que el indicador no sirva para backtest así como está planteado.

La idea de restar medias móviles es reducir el ruido pero como desventaja da una medida referente al momento que no sirve para comparar contra momentos en los que el precio es muy diferente, vean el ejemplo práctico:

Tesla, Inc. - 1D - Cboe BZX = 0732.25 H734.00 L724.20 C733.57 +1.18 (+0.16%)

729.88 15.12 740.00

Vean como el indicador MACD parece muerto antes de 2020

Esto se debe a esa construcción del indicador como una resta de medias móviles, lo que lo hace inservible, así como está para backtestear ideas de trading. Ahora me dirán, momento, podemos usar de señal cuando el histograma pasa de positivo a negativo o algo similar, que solo vea el signo, y no estaría mal el planteo, pero es muy pobre, perdemos toda la riqueza de una serie de precios para una señal binaria tipo "+" o "-".



Como veremos en solo un par de páginas, la solución para poder enriquecer este tipo de indicadores podría venir por transformarlo en un oscilador acotado. De hecho voy a usar estas mismas funciones para exemplificar ese procedimiento de generar un oscilador de cualquier indicador.

Bueno, veamos entonces la construcción del indicador en Python, es muy sencilla

Lo planteamos directamente como una función y buscamos la data de yfinance adentro de la función misma

```
def macd(ticker, slow, fast, suavizado):
    data = yf.download(ticker, auto_adjust=True)
    data['ema_fast'] = data.Close.ewm(span=fast).mean()
    data['ema_slow'] = data.Close.ewm(span=slow).mean()
    data['macd'] = data.ema_fast - data.ema_slow
    data['signal'] = data.macd.ewm(span=suavizado).mean()
    data['histograma'] = data.macd - data.signal
    data = data.dropna().round(2)
    return data
```

Probamos nuestra función:

```
macd('TSLA', 26, 12, 9)
```

Date	Open	High	Low	Close	Volume	ema_fast	ema_slow	macd	signal	Histograma
2010-06-23	3.80	5.00	3.51	4.73	20221000	4.73	4.73	0.00	0.00	0.00
2010-06-30	5.10	6.58	4.65	4.77	20206000	4.77	4.77	-0.00	-0.00	-0.00
2010-07-01	5.00	5.18	4.65	4.58	41124000	4.52	4.54	-0.01	-0.00	-0.01
2010-07-02	4.80	4.82	3.76	3.81	20499000	4.38	4.41	-0.04	-0.02	-0.02
...
2021-08-31	733.00	740.39	728.44	735.72	20805400	739.43	887.23	12.13	3.80	2.34
2021-09-01	734.08	741.98	731.27	734.06	13224000	713.22	700.32	13.20	10.48	2.72
2021-09-02	734.50	745.57	730.54	732.38	12777300	716.17	712.42	13.25	11.73	2.52
2021-09-03	732.25	734.00	724.20	733.57	13244100	718.85	716.72	14.12	11.73	2.39

2817 rows × 10 columns

Siempre que armen el formulario para un indicador, verifiquen que les da bien, a veces por mas simple que parezca uno se equivoca y después está todo el backtest basado en un indicador mal construido. Dicho esto, vean en el gráfico de la página anterior que me dan bien los valores para hoy 3 de septiembre el histograma 2.39 etc... pero verifiquemos con la API de alphavantage, ¿se acuerdan del t[3]? Bueno, esta API tiene todos los indicadores técnicos así que está buena para chequear que lo están construyendo bien.

```
def macd_API(symbol, interval, series_type='close', fastperiod=12, slowperiod=26, signalperiod=9):
    function='MACD'
    apikey = 'N9P0qSV7x9VSJXY4'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval':interval,
                  'series_type':series_type, 'fastperiod': fastperiod,
                  'slowperiod':slowperiod, 'signalperiod':signalperiod,
                  'apikey':apikey }

    r = requests.get(url, params=parametros)
    try:
        js = r.json()['Technical Analysis: MACD']
        df = pd.DataFrame.from_dict(js, orient='index')
        df = df.astype('float')
        df.index.name = 'Date'
        df = df.sort_values('Date', ascending=True).round(2)
        df.index = pd.to_datetime(df.index)
    except:
        df = pd.DataFrame()
        print(r.json())
    return df

data = macd_API('TSLA','daily')
data
```

MACD_Hist MACD MACD_Signal

Date	MACD_Hist	MACD	MACD_Signal
2010-08-16	-0.09	-0.05	0.04
2010-08-17	-0.07	-0.05	0.02
2010-08-18	-0.06	-0.05	0.01
2010-08-19	-0.05	-0.06	-0.01
...
2021-08-31	2.34	12.13	9.80
2021-09-01	2.72	13.20	10.48
2021-09-02	2.62	13.75	11.13
2021-09-03	2.39	14.12	11.73

2784 rows × 3 columns

Como comparación veamos el famoso RSI que si es un oscilador acotado, es decir varía entre 0 y 100, independientemente del nivel del precio, otra ejemplo que me gusta por lo sencillo para comprender es el AROON que es un oscilador acotado pero partiendo de un conteo discreto, luego veremos mejor este tipo de construcciones así que lo dejo para después, volvamos al RSI, es una construcción basada en un agrupamiento, es decir separar las velas alcistas de las bajistas, y luego contrastar una contra otra en un ratio acotado.

Es muy importante comprender los indicadores y como están construidos para entender que nos están indicando y poder diseñar un modelo de trading coherente.

En este caso la construcción del RSI es la siguiente:

- Se elige una ventana de velas hacia atrás "n"
- Se separan los movimientos de las "n" velas en las alcistas y las bajistas
- Se calcula una media móvil exponencial de cada grupo con factor de decaimiento $\alpha = 1/n$
- Se calcula un ratio de ambas EMAs llamado "rs" como $EMA_{bulls} / EMA_{bears}$
- Se acota el oscilador entre 0 y 100 con la fórmula: $RSI = 100 - (100 / (1+rs))$

Como ven el rs oscila entre 0 e infinito, no es acotado, ya que tenderá a 0 cuando sean todas las velas bajistas y tenderá a infinito cuando sean todas alcistas, ahora al hacer $100/(1+rs)$ el denominador cuando valga infinito hará 0 ese cociente, y cuando valga cero lo hará 100, con lo cual se termina acotando el oscilador entre 0 y 100. Hay mil formas de armar un oscilador acotado, esta es una, pero veremos otras más generales luego.

Veamos esto en código entonces:

```
import yfinance as yf
import numpy as np

data = yf.download('TSLA', end='2021-09-04', auto_adjust=True)

ruedas = 14

data['dif'] = data['Close'].diff()
data['win'] = np.where(data['dif'] > 0, data['dif'], 0)
data['loss'] = np.where(data['dif'] < 0, abs(data['dif']), 0)

data['ema_win'] = data.win.ewm(alpha=1/ruedas).mean()
data['ema_loss'] = data.loss.ewm(alpha=1/ruedas).mean()
data['rs'] = data.ema_win / data.ema_loss
data['rsi'] = 100 - (100 / (1+data.rs))
data = data.reset_index().dropna().round(2)
data
```

	Date	Open	High	Low	Close	Volume	dif	win	loss	ema_win	ema_loss	rs	rsl
1	2010-06-30	5.16	6.08	4.66	4.77	85935500	-0.01	0.00	0.01	0.00	0.01	0.00	0.00
2	2010-07-01	5.00	5.18	4.05	4.39	41094000	-0.37	0.00	0.37	0.00	0.14	0.00	0.00
3	2010-07-02	4.60	4.62	3.74	3.84	25699000	-0.55	0.00	0.55	0.00	0.25	0.00	0.00
4	2010-07-06	4.00	4.00	3.17	3.22	34334500	-0.62	0.00	0.62	0.00	0.34	0.00	0.00
5	2010-07-07	3.28	3.33	3.00	3.16	34608500	-0.06	0.00	0.06	0.00	0.28	0.00	0.00
...
2812	2021-08-30	714.72	731.00	712.73	730.91	18604200	18.99	18.99	0.00	7.53	4.65	1.62	61.80
2813	2021-08-31	733.00	740.39	726.44	735.72	20855400	4.81	4.81	0.00	7.34	4.32	1.70	62.93
2814	2021-09-01	734.08	741.99	731.27	734.09	13204300	-1.63	0.00	1.63	6.81	4.13	1.65	62.26
2815	2021-09-02	734.50	740.97	730.54	732.39	12777300	-1.70	0.00	1.70	6.33	3.96	1.60	61.52
2816	2021-09-03	732.25	734.00	724.20	733.57	15246100	1.18	1.18	0.00	5.96	3.67	1.62	61.86

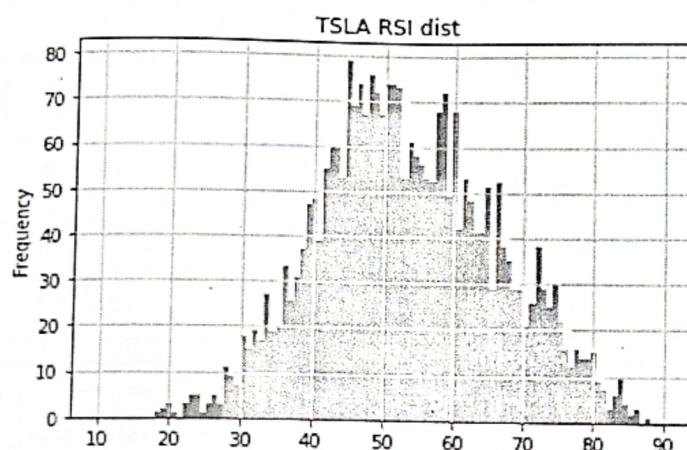
2816 rows × 13 columns

Una de las enormes ventajas de los osciladores acotados, es que me permiten comparar un dato de hoy con un dato de cuando el activo valía algo que nada que ver, ya que el indicador es agnóstico al precio, esto para un backtesting es algo fundamentalísimo, ya que necesito uniformidad en los indicadores a usar.

Entonces ya que tengo la ventaja de tener el oscilador acotado, puedo construir un histograma del mismo, lo cual es algo super útil para entender mejor el funcionamiento del indicador, cosa que por ejemplo con el MACD no podía hacer ya que era relativo al precio.

Veamos entonces como queda ese histograma con TSLA

```
data.rsi.plot(kind='hist', grid=True, title='TSLA RSI dist', bins=100, range=(10,90))  
plt.show()
```



Toda la pinta de una distribución normal centrada, pero corroboremos

```
from scipy import stats  
  
print(f"La kurtosis es {data.rsi.kurt():.2f}")  
print(f"El skew es {data.rsi.skew():.2f}")  
print(f"La media es {data.rsi.mean():.2f}")  
print(f"El desvío standar es {data.rsi.std():.2f}")  
print(f"La proba que sea una dist normal es: {stats.normaltest(data.rsi).pvalue:.2%}")
```

La kurtosis es 0.01
El skew es 0.06
La media es 53.44
El desvío standar es 12.86
La proba que sea una dist normal es: 41.28%

Evidentemente un pvalor > 0.1 ya me dice que no puedo rechazar la hipótesis nula del test de hipótesis, que asumía distribución normal, con lo cual la acepto digamos

O sea que en el caso de TSLA el RSI de media dio 53.44, es decir estamos en presencia de mas velas alcistas que bajistas, ya que el RSI tiende a 100 cuando es 100% alcista y a 0 cuando es 100% bajista, y lo mas rico es que si el desvío estándar es 12.86 y sabemos que es una dist normal, entonces:

$$\begin{aligned} \Pr(\mu - 1\sigma \leq X \leq \mu + 1\sigma) &\approx 68.27\% \\ \Pr(\mu - 2\sigma \leq X \leq \mu + 2\sigma) &\approx 95.45\% \\ \Pr(\mu - 3\sigma \leq X \leq \mu + 3\sigma) &\approx 99.73\% \end{aligned}$$

O sea:

$\mu \pm 2\sigma \Rightarrow 53.44 \pm 2 \cdot 12.86 / 53.44 + 2 \cdot 12.86$ o sea entre 27.72 y 79.16 (El RSI de TSLA está el 95.45% de las ruedas entre 27.72 y 79.16, con lo cual por debajo de ese rango está super bearish y por encima super bullish)

Muchos libros ponen zona de sobre compra (ultra bullish) al 70 y zona de sobre venta (ultra-bearish) el 30. Acá nos da 27.7 y 79.2 redondeando, es decir, parecido, pero lo importante es entender el concepto del indicador, al ser un oscilador que sigue una normal, lo que nos muestra es que fuera de ese rango de 2 sigma, hay solo un 5% de probabilidad de mantenerse, con lo cual, es mucho mas probable que corrija a que se mantenga así, con lo cual sería un buen indicador del tipo "contrarian" aunque muchas veces se lo usa como del tipo *tren following* adentro de ese +/- 2sigmas

El RSI en Python entonces con un gráfico para visualizar:

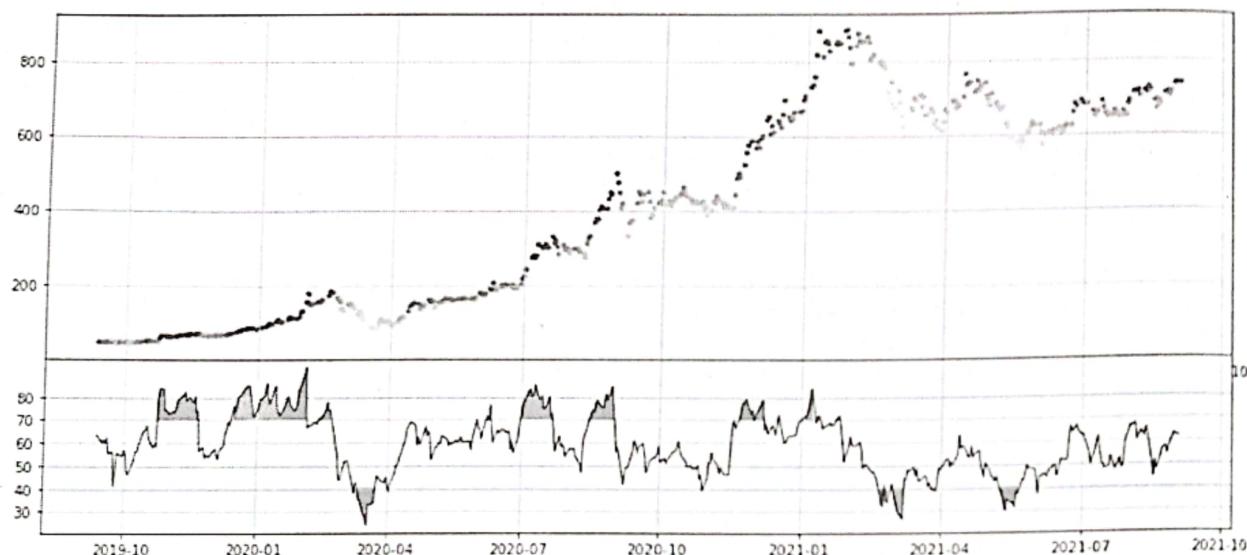
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(15,5), nrows=2, gridspec_kw={'height_ratios': [2, 1]})

df = data.iloc[-500:]
ax[0].scatter(df.index, df.Close, s=10, cmap="binary", c=df.rsi/100)
ax[0].grid()

ax[1].plot(df.index, df.rsi, lw=1c='k')
ax[1].set_yticks(range(10,90,10))

ax[1].fill_between(df.index, df.rsi, 70, where=df.rsi>70, alpha=0.5, color='k')
ax[1].fill_between(df.index, df.rsi, 40, where=df.rsi<40, alpha=0.5, color='k')
ax[1].grid()
plt.subplots_adjust(hspace=0)
```



Como ven el problema de usar este indicador como contrarian para esperar la corrección en sentido inverso, es que puede mantenerse "sobrecomprado" o "sobrevendido" con el RSI por fuera de los 2 sigma, muchas velas, con lo cual es riesgoso usarlo así nomás, por ello se suele usar en esas zonas fuera de los 2 sigma, como indicador de divergencia, pero ya es otro tema, no me quiero desviar mucho, el punto por ahora es estudiar la "estructura" de los indicadores, entenderla y con ello luego vamos a usar de materia prima para hacer backtests de los sistemas de trading.

Entonces resumiendo este oscilador acotado tiene 2 tipos de zonas

- Dentro de la banda +/- 2 sigma: Movimientos típicos, puede usarse como tren following, se consideraría alcista la tendencia arriba del "mu" en este caso de TSLA 53.44
- Fuera de la banda: Se consideraría saturado esperando corrección en sentido inverso es decir arriba de 2sigma de la media se esperaría pronto retroceso, debajo de 2sigma de la media se esperaría pronto rebote.

Bueno, por último, me faltaba hablarles de algún indicador meramente contraria, que se yo, por agarrar alguno vamos a usar el AROON, me resulta piola para mostrar algo diverso, pero además me encanta porque me ha dado grandes satisfacciones en lo personal

Veamos, es un indicador construido con un conteo discreto, ya veremos mas adelante este tipo de indicadores, pero como es super conocido lo pongo en este repaso, se trata de contabilizar (dentro de una ventana de "n" velas) cuantas velas hace que marcó mínimo y máximo y sacar el % de la ventana que se dieron ambos eventos

Veamos esto en código:

```
import yfinance as yf
import numpy as np

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04')
n=25

data['smin'] = data.Low.shift().rolling(n).apply(lambda x: x.argmin())
data['smax'] = data.High.shift().rolling(n).apply(lambda x: x.argmax())
data['aroon_up'] = 100* data['smax'] / n
data['aroon_down'] = 100* data['smin'] / n

tail = data.tail(25)
tail.style.highlight_max('High').highlight_min('Low')
```

Date	Open	High	Low	Close	Volume	smin	smax	aroon_up	aroon_down
2021-08-02 00:00:00	700.000000	726.940002	698.400024	709.669983	33615800	8.000000	5.000000	20.000000	32.000000
2021-08-03 00:00:00	719.000000	722.650024	701.010010	709.739990	21620300	7.000000	24.000000	96.000000	28.000000
2021-08-04 00:00:00	711.000000	724.900024	708.929993	710.919983	17002600	6.000000	23.000000	92.000000	24.000000
2021-08-05 00:00:00	716.000000	720.950012	711.409973	714.630005	12919600	5.000000	22.000000	88.000000	20.000000
2021-08-06 00:00:00	711.900024	716.330017	697.630005	699.099978	15576200	4.000000	21.000000	84.000000	16.000000
2021-08-09 00:00:00	710.169983	719.030029	705.130005	713.760010	14715300	3.000000	20.000000	80.000000	12.000000
2021-08-10 00:00:00	713.989990	716.590027	701.880005	709.989990	13432300	2.000000	19.000000	76.000000	8.000000
2021-08-11 00:00:00	712.710022	715.179993	704.210022	707.820007	9800600	1.000000	18.000000	72.000000	4.000000
2021-08-12 00:00:00	706.340027	722.789988	699.400024	722.250000	17459100	0.000000	17.000000	68.000000	0.000000
2021-08-13 00:00:00	723.710022	729.900024	714.340027	717.169983	16698900	6.000000	16.000000	64.000000	24.000000
2021-08-16 00:00:00	705.070007	709.500000	676.400024	686.169983	22677400	5.000000	24.000000	96.000000	20.000000
2021-08-17 00:00:00	672.659973	674.580017	648.840027	665.710022	23721300	4.000000	23.000000	92.000000	16.000000
2021-08-18 00:00:00	669.750000	695.770020	669.349976	688.989900	20349400	3.000000	22.000000	88.000000	12.000000
2021-08-19 00:00:00	678.210022	686.549988	667.580027	673.469971	14313500	2.000000	21.000000	84.000000	8.000000
2021-08-20 00:00:00	682.849976	692.130005	673.700012	680.260010	14781800	1.000000	20.000000	80.000000	4.000000
2021-08-23 00:00:00	685.440002	712.130005	680.750000	706.299988	20254900	0.000000	19.000000	76.000000	0.000000
2021-08-24 00:00:00	710.679993	715.219971	702.640015	708.489900	13083100	5.000000	18.000000	72.000000	20.000000
2021-08-25 00:00:00	707.030029	716.969971	704.000000	711.200012	12645600	4.000000	17.000000	68.000000	16.000000
2021-08-26 00:00:00	708.309998	715.400024	697.619995	701.159973	13214300	3.000000	16.000000	64.000000	12.000000
2021-08-27 00:00:00	705.000000	715.000000	702.099976	711.919983	13762100	2.000000	15.000000	60.000000	8.000000
2021-08-30 00:00:00	714.719971	731.000000	712.729980	730.909973	18604200	1.000000	14.000000	56.000000	4.000000
2021-08-31 00:00:00	733.000000	740.390015	726.440002	735.719971	20855400	0.000000	24.000000	96.000000	0.000000
2021-09-01 00:00:00	734.080017	741.989990	731.270020	734.090027	13204300	0.000000	24.000000	96.000000	0.000000
2021-09-02 00:00:00	734.500000	740.969971	730.539978	732.390015	12777300	0.000000	24.000000	96.000000	0.000000
2021-09-03 00:00:00	732.250000	734.000000	724.200012	733.570007	15246100	12.000000	23.000000	92.000000	48.000000

Como ven, de los últimos 25 velas, el mayor máximo se dio en la 23/25 (92%) y el menor mínimo en la 12/25 (48%), esos valores 92 y 48 son las aroon up y down respectivamente. La idea es que cuando el Aroon Up cruza encima del Aroon down esta por comenzar tendencia alcista y viceversa. Intenta anticipar cambio de tendencia.

Como ven es un oscilador acotado por definición entre 0 y 100%, intenta anticipar cambios porque no se basa en medias que atrasan por arrastre estadístico una cantidad "n" de valores, sino que toma el máximo o mínimo de la serie, con lo cual un solo valor ya modifica al indicador

Igual insisto en algo super importante, es irrelevante lo que lean en los libros de "como se usa un indicador", justamente porque lo que nos va a interesar acá es backtestear esto y corroborarlo empíricamente

Empecemos por graficarlo para visualizar la idea:

```
import yfinance as yf
import matplotlib.pyplot as plt

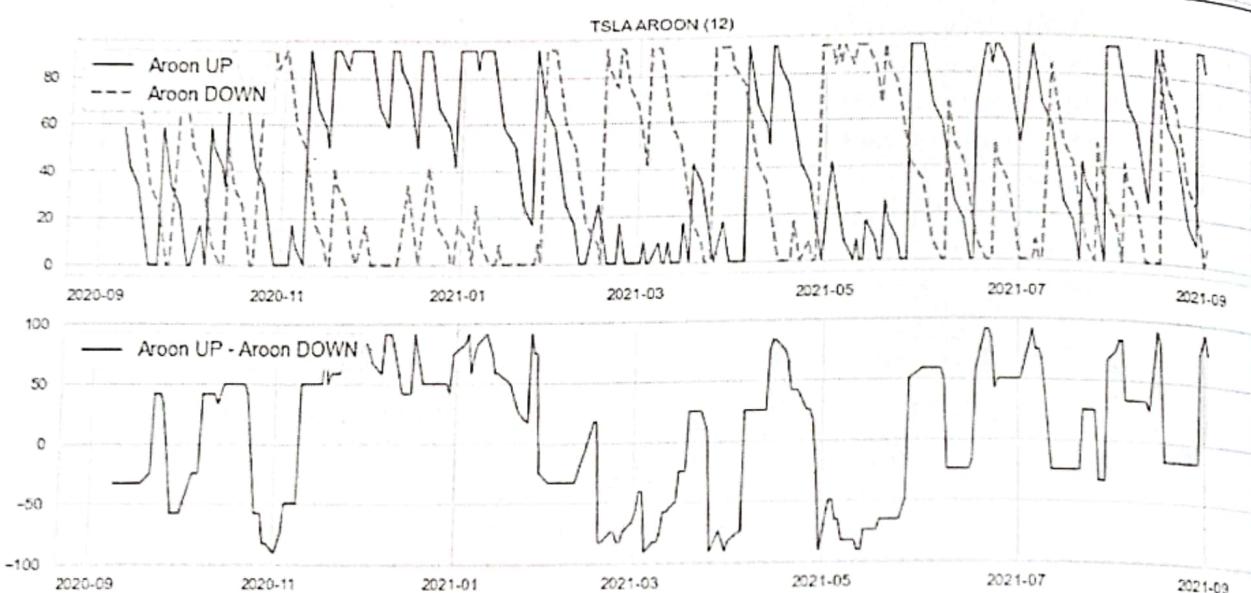
data = yf.download('TSLA', auto_adjust=True, end='2021-09-04')
n=12

data['smin'] = data.Low.shift().rolling(n).apply(lambda x: x.argmin())
data['smax'] = data.High.shift().rolling(n).apply(lambda x: x.argmax())
data['aroon_up'] = 100* data['smax'] /n
data['aroon_down'] = 100* data['smin'] /n
data['aroon_dif'] = data.aoon_up - data.aoon_down

last_year = data.iloc[-250:,-3:]

fig, ax = plt.subplots(figsize=(15,7), nrows=2)
ax[0].set_title(f'TSLA AROON ({n})')
ax[0].plot(last_year['aroon_up'], lw=1, ls='-', color='k', label='Aroon UP')
ax[0].plot(last_year['aroon_down'], lw=2, ls='--', color='gray', label='Aroon DOWN')
ax[0].legend(loc='upper left', fontsize=14)

ax[1].plot(last_year['aroon_dif'], lw=1, color='k', label='Aroon UP - Aroon DOWN')
ax[1].fill_between(last_year.index, last_year['aroon_dif'], 0, where=last_year['aroon_dif']>0, alpha=0.2)
ax[1].fill_between(last_year.index, last_year['aroon_dif'], 0, where=last_year['aroon_dif']<0, alpha=0.2)
ax[1].legend(loc='upper left', fontsize=14)
```



Como ven al ser un indicador discreto basado en un solo máximo o mínimo, da saltos bruscos, por eso es que se suele usar de gatillo para anticipar un potencial movimiento direccional o inicio de nueva tendencia o cambio de la tendencia previa. Pero si bien aun falta mucho para poder backtestear esto, empecemos por lo que vimos de data-mining básico, es decir, probemos así nomás, así como lo tenemos, si me correlaciona algo o nada con algún forward o retornos futuros a "x" velas adelante.

Para ello usamos el mismo código que usamos antes para probar con el cruce de medias y el RSI y la volatilidad, veremos los r^2 y un histograma según la clasificación exposit del forward y un scatter de correlación.

Arrancamos calculando varios forwards, elegimos rango de 10 a 20 velas, dado que configuramos el indicador con una ventana previa de 12 velas, parece un rango sensato;

```
import yfinance as yf  
data = yf.download('TSLA', auto_adjust=True, end='2021-09-24')  
n=12  
  
data['smin'] = data.Low.shift().rolling(n).apply(lambda x: x.argmax())  
data['smax'] = data.High.shift().rolling(n).apply(lambda x: x.argmax())  
data['aroon_up'] = 100* data['smax'] / n  
data['aroon_down'] = 100* data['smin'] / n  
  
data['aroon_dif'] = data.aroon_up - data.aroon_down  
  
rango_fw = range(10,20)  
for i in rango_fw:  
    data[f'fw_{i}'] = (data.Close.shift(-i)/data.Close-1) * 100  
  
cols = ['aroon_dif'] + [f'fw_{i}' for i in rango_fw]  
data = data.loc[:,cols].dropna().round(4)  
data
```

Date	aroon_dif	fw_10	fw_11	fw_12	fw_13	fw_14	fw_15	fw_16	fw_17	fw_18	fw_19
2010-07-16	-33.3333	-3.3915	1.3568	6.3469	3.0039	-0.9205	-5.0872	-5.0388	-7.8004	-13.2752	-14.7287
2010-07-19	-33.3333	-4.5185	0.1228	-2.9667	-6.6638	-10.5888	-10.5431	-13.1447	-18.3021	-19.6714	-16.3852
2010-07-20	-25.0000	8.1281	4.7231	0.7329	-3.4975	-3.4483	-6.2562	-11.8227	-13.3005	-9.7527	-7.4877
2010-07-21	-16.6667	5.1434	1.1375	-3.1157	-3.0663	-5.8853	-11.4738	-12.9575	-9.3866	-7.1217	-5.2918
2010-07-22	66.6667	-2.6190	-8.7143	-8.6667	-9.3809	-14.7619	-16.1905	-12.7619	-10.5714	-8.8095	-10.6190
...
2021-08-03	83.3333	-6.2037	-2.9238	-5.1103	-4.1536	-0.4847	-0.1761	0.2057	-1.2089	0.3072	2.9828
2021-08-04	83.3333	-3.0847	-5.2678	-4.3127	-0.6499	-0.3418	0.0394	-1.3729	0.1407	2.8118	3.4384
2021-08-05	83.3333	-5.7598	-4.8095	-1.1656	-0.8592	-0.4800	-1.8849	-0.3792	2.2781	2.9512	2.7231
2021-08-06	83.3333	-2.6949	1.0299	1.3432	1.7308	0.2947	1.8338	4.5501	5.2382	5.0050	4.7618
2021-08-09	83.3333	-1.0452	-0.7383	-0.3587	-1.7653	-0.2578	2.4028	3.0767	2.8483	2.6101	2.7754

2786 rows × 11 columns

Vemos uno por uno los valores de r^2

```
fws = [f'fw_{i}' for i in rango_fw]  
  
for fw in fws:  
    r2 = data['aroon_dif'].corr(data[fw])  
    print(f"El r2 con {fw} es {r2:.4f}")
```

El r2 con fw_10 es 0.0834
El r2 con fw_11 es 0.0842
El r2 con fw_12 es 0.0891
El r2 con fw_13 es 0.0918
El r2 con fw_14 es 0.0967
El r2 con fw_15 es 0.0992
El r2 con fw_16 es 0.0988
El r2 con fw_17 es 0.0981
El r2 con fw_18 es 0.0948
El r2 con fw_19 es 0.0889

Como ven bastante parejo pero encuentra un pico de correlación en un fw de 15 velas adelante

Bien, entonces visualicemos esto en una clasificación ex post del fw:

Elegimos el fw_15 porque es el que mas correlación marcó

Primero generamos la columna de clasificación y calculamos la media de cada grupo y desvio:

```
data['variacion_tipo'] = np.where(data.fw_15 > 0, 'Up', 'Down')
medias = data.groupby('variacion_tipo').mean().loc[:, features].round(2)
desvios = data.groupby('variacion_tipo').std().loc[:, features].round(2)
print(f'Medias según clasificación de Variación \n {medias} \n')
print(f'Desvíos según clasificación de Variación \n {desvios} \n')
```

Medias según clasificación de Variación

variacion_tipo	aroon_dif
Down	7.47
Up	12.66

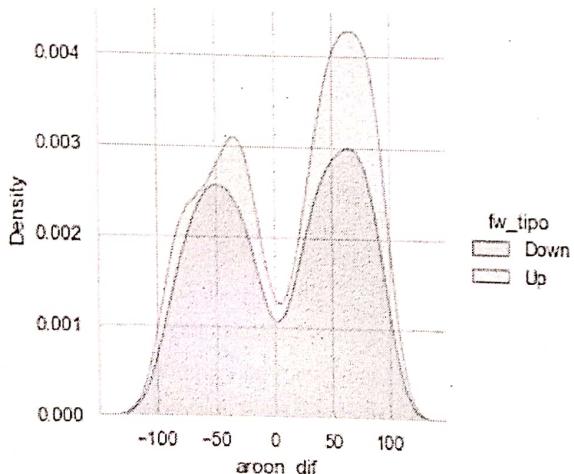
Desvíos según clasificación de Variación

variacion_tipo	aroon_dif
Down	58.02
Up	58.76

Como ven, si bien hay una clara diferencia en las medias, los desvíos son muy grandes

Ambas arriba de 0 la diferencia de aroon_up - aroon_down, pero el desvío es muchísimo mas grande que la diferencia de 5 puntos porcentuales en las medias, así que ya me indica que no es gran cosa lo que me informa en términos agregados esta clasificación, pero veámoslo con los KDEs

```
import seaborn as sns
data['fw_tipo'] = np.where(data.fw_15 > 0, 'Up', 'Down')
sns.displot(data, x="aroon_dif", hue="fw_tipo", kind="kde", height=4, fill=True)
```



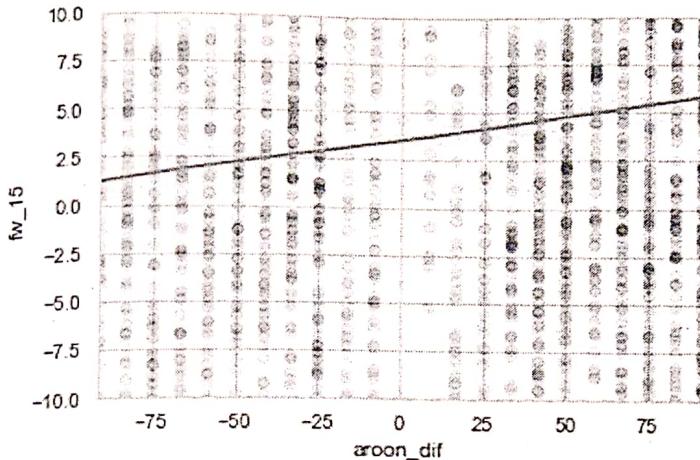
Lo dicho, corroboramos que a nivel agregado no me indica mucho esta clasificación, ahora, esto no significa que el indicador, no sirve de nada, simplemente no es un buen generalizador de una situación futura, pero como les digo al ser un indicador hiper sensible a un solo dato de la serie puede ser muy bueno como gatillo tanto en una entrada como en una salida y sobre todo estas últimas en mi experiencia.

Bueno, vemos por último un gráfico de correlación para completar el análisis con una recta de regresión

```
import seaborn as sns
sns.set(style = 'whitegrid')

g = sns.regplot(x="aeroon_dif", y="fw_15", data=data, line_kws={"color": "red"}, scatter_kws={"color": "k", "alpha":.1})

g.set(ylim=(-10, 10))
```



Si bien ya sabíamos que el indicador en términos agregados no era gran cosa, en este caso visualizamos como viene dada esa dispersión y queda un poco mas claro ahora que con el histograma, hay veces que pasa esto que un tipo de gráfico es mas claro que otra en un ejemplo y pasa lo contrario con otro ejemplo u otro indicador, por eso les muestro los que a mi me resultan más útiles.

Indicadores de flujos (volumen y derivados)

Bueno, siguiendo con el plan de repasar el top10 de los tipos de indicadores mas usuales para backtesting, vamos con los de flujos, generalmente basados en cuestiones de volumen y derivados pero también podrían basarse en alguna métrica de flujo en sí, por ejemplo en el mercado crypto muchos usan indicadores del stock de BTC en las wallets exchanges centralizados, entendiendo que el flujo de wallets externas a los exchanges es presión de venta de BTC o bajista al precio, mientras que el flujo de los exchanges a wallets externas es reducción de oferta y por ende presión alcista al precio.

Por otro lado también hay métricas basadas en derivados, volviendo el tema cryptos, se habla mucho del grado de apalancamiento del mercado, como es un mercado ultra volátil, el grado de apalancamiento muchas veces es extremo, con lo cual muchos miden este factor para intentar predecir momentos de saturación de los ciclos alcistas o bajistas con métricas como la tasa de fondeo de los futuros apalancados, o el ratio put/call de las opciones por mencionar algunas métricas

Finalmente están los típicos indicadores que combinan la dirección de las velas con la serie de volúmenes por vela para formar indicadores de flujo basados simplemente en la serie OHLC+Volume

Entonces podríamos decir que a su vez una clasificación de indicadores de flujos de dinero podría subdividirse en las siguientes tres categorías principales

- Construcciones sobre la serie de volumen negociado
 - o VWAP (Precio ponderado por volumen)
 - o OBV (On Balance Volume, acumulador del volumen, suma/testa volumen según el precio)
 - o AD (Chaikin Acumulación/Distribución, una mejora del OBV con una ponderación de la forma de la vela)
 - o ADOSC (Oscilador del Chaikin AD)
- Construcciones sobre derivados (futuros y opciones)
 - o Ratio put/call de opciones
 - o Tasa de fondeo de futuros
 - o Open interest en futuros
 - o Open interest en opciones
- Métricas externas de flujo (ejemplos del mercado crypto)
 - o Movimientos extra-exchanges
 - o Movimientos intra-exchanges
 - o Minteo/Quema y series de stocks de stablecoins

En el tercer caso les pongo ejemplos del mercado crypto porque al ser un mercado mucho más transparente en el sentido de acceso a datos e información, son métricas que googleando un poco van a poder conseguir en diversas APIs totalmente gratis.

Bueno, veamos alguno, tomo el OBV como base para explicar cómo acotar un oscilador

El OnBalance Volume (OBV) es el indicador de flujo más básico que existe, se trata de ir acumulando el volumen "nominal" cuando el precio sube y restando cuando baja, vela a vela, de esa forma se obtiene un saldo neto acumulado de flujo, veámoslo en Python

```
import yfinance as yf
import numpy as np

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04', progress=False)

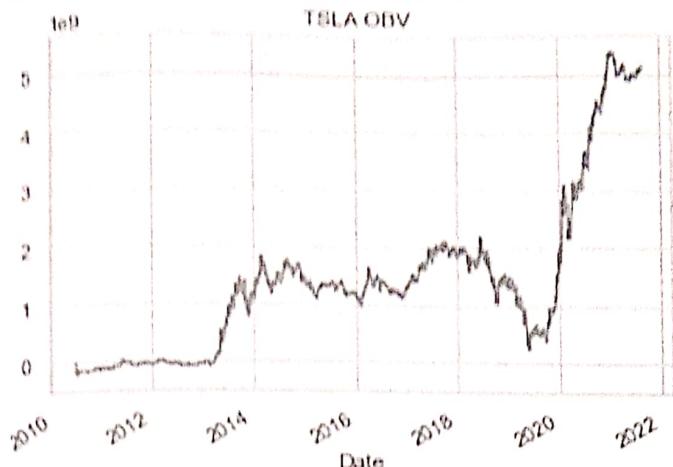
data['Balance'] = np.where(data.Close>data.Close.shift(), data['Volume'],
                           np.where(data.Close < data.Close.shift(), -data['Volume'], 0))
data['OBV'] = data['Balance'].cumsum()
data
```

Date	Open	High	Low	Close	Volume	Balance	OBV
2010-06-29	3.800000	5.000000	3.505000	4.778000	93831500	0	0
2010-06-30	5.158000	6.084000	4.660000	4.766000	85935500	-85935500	-85935500
2010-07-01	5.000000	5.184000	4.054000	4.392000	41094000	-41094000	-127029500
2010-07-02	4.600000	4.620000	3.742000	3.840000	25699000	-25699000	-152728500
2010-07-06	4.000000	4.000000	3.165000	3.222000	34334500	-34334500	-187063000
...
2021-06-30	714.719971	731.000000	712.729980	730.909973	18604200	18604200	5107400000
2021-08-31	733.000000	740.390015	726.440002	735.719971	20855400	20855400	5128255400
2021-09-01	734.080017	741.989990	731.270020	734.090027	13204300	-13204300	5115051100
2021-09-02	734.500000	740.969971	730.539978	732.390015	12777300	-12777300	5102273800
2021-09-03	732.250000	734.000000	724.200012	733.570007	15246100	15246100	5117519900

2817 rows x 7 columns

Como muchos se imaginarán el problema con este indicador así como está es bastante evidente:

```
data.ohv.plot()
```



El indicador es relativo al momento, es decir, que no puedo comparar lo que vale en 2021, con lo que media en 2012 porque es un acumulado total.

Primera idea de customización de indicadores para backtesting:

Homogeneizar toda la serie de datos a una escala acotada desestacionalizada

Perdón que voy a hacer un paréntesis acá en el medio de este título 2 de 10 de los tipos de indicadores para explicar como lograr la construcción de un oscilador a partir de cualquier otro indicador técnico

Hago esto ahora porque recuerdo un trader que me dijo que era una lástima que el indicador que mas conocía y le servía mucho, no lo podía usar para backtesting porque tenía esa desventaja de no ser acotado, obviamente le comenté que no normalice o lo estandarice de alguna forma, así que acá estoy volcando un poco eso en este tomo

Así que nada, "título colado" en este top10 de tipos de indicadores, luego seguimos con el resto y con lo que me faltaba del N°2 "Indicadores de Flujos"

Construcción de Osciladores acotados desde cualquier indicador

Suena como medio raro toto, pero es ni mas ni menos que llevar este indicador a algo que sin importar el año en que lo mida, pueda obtener un rango similar, y luego acotarlo, pero vamos por partes, primero llevar a algo que mas o menos me pueda dar lo mismo en cualquier año que lo mida

Tomo de ejemplo para convertir en oscilador al OBV solo porque me pareció claro para visualizar el problema a solucionar

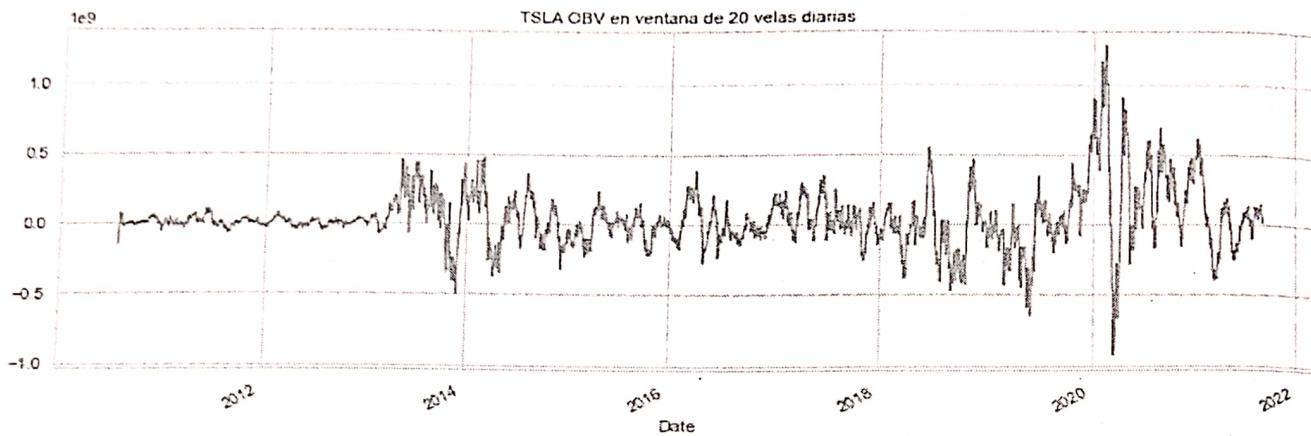
Para ello, simplemente debo transformar el OBV acumulado total en uno de una ventana razonable de velas, por ejemplo, 20 velas, veamos que me da:

```
import yfinance as yf
import numpy as np

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04', progress=False)
n = 20

data['Balance'] = np.where(data.Close>data.Close.shift(), data['Volume'],
                           np.where(data.Close < data.Close.shift(), -data['Volume'], 0))
data['OBV'] = data['Balance'].rolling(n).sum()
titulo = 'TSLA OBV en ventana de 20 velas diarias'

data.OBV.plot(figsize=(15,5), title=titulo)
```



Va queriendo ¿no?

Bueno, al menos ahora solo arrastra 20 velas con lo cual tranquilamente podría obtener en 20 días un valor similar a uno de 2012, pero sigue estando fuera de cota, es decir, miren esos valores entre -1billon y 1 billón de nominales, el día de mañana aparece un volumen extraordinario en una ventana de 20 días, y me deja todo el resto fuera de escala otra vez, de hecho vean como en 2020 con el evento de la crisis covid se descontroló bastante

Bueno, viene el momento de acotar esto en un rango definido, generalmente se usa la escala 0,1 o bien 0,100, pero también es válida -1,1, etc.. La idea es que, pase lo que pase, siempre estemos en una escala acotada entre un máximo absoluto fijo y un mínimo absoluto fijo

¿para que quiero eso?

Y para backtestear, simplemente necesito comparar peras con peras, después tendremos si quieren millones de discusión metodológicas de si está bien o mal tal o cual cosa, es todo un mundo este tema, pero necesitamos partir de bases mínimas (necesarias, no suficientes si quieren)

Volviendo, vamos a acotar esto, hay 3 formas típicas

- Por distancia al mínimo dentro del rango, dentro de una ventana de "n" valores
- Por zscores, dentro de una ventana de "n" valores
- Por zscore de la serie completa (ojo con la trampa de usar datos futuros "**Look-ahead.Bias**")
- Por percentil (ojo con trampa de usar datos futuros "**Look-ahead.Bias**")

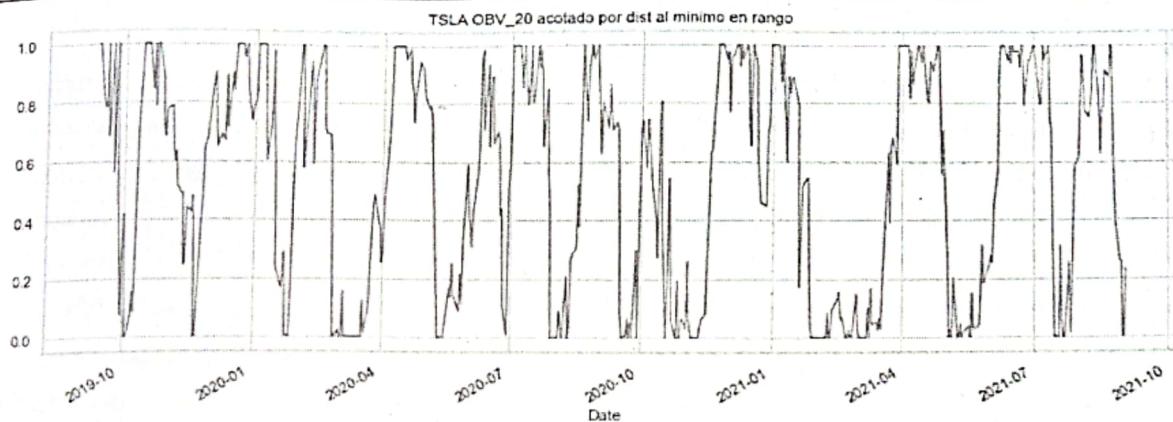
Por distancia al mínimo dentro del rango sería hacer algo como:

$$y_i = \frac{x_i - \min(X)}{\max(X) - \min(X)}$$

Siendo Y el vector acotado de X, o sea cada y_i valor acotado de x_i para todos los "i" de la serie

Veámoslo en código (ploteo solo los últimos 2 años para que se entienda mejor):

```
data['OBV_acotado'] = (data.OBV - data.OBV.rolling(n).min()) / (data.OBV.rolling(n).max() - data.OBV.rolling(n).min())
data['OBV_acotado'].iloc[-500:].plot(figsize=(15,5), title=f'TSLA OBV_{n} acotado por dist al minimo en rango')
```



Está bueno porque queda acotado entre 0 y 1 y es sensible a la ventana, esta ventana "n" (que tenía seteada en 20) pasa a ser un hiperparámetro de nuestro backtesting, o sea un parámetro optimizable.

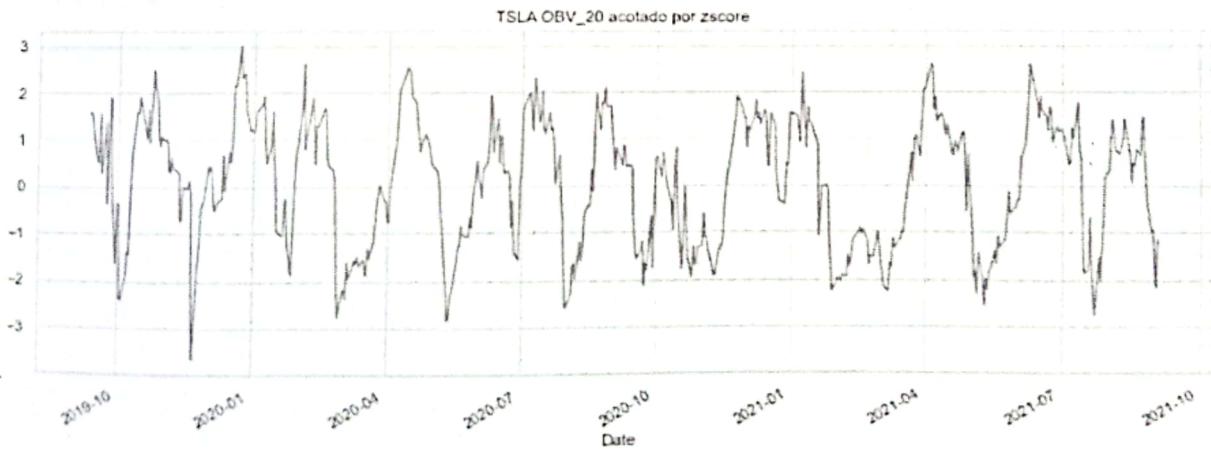
La 2º forma es por zscores, es decir por distancias "en sigmas" a la media, o sea expresar cada OBV por cuantos desvíos estándar está alejado para arriba o para debajo de la media de todos los valores de la serie.

Si llamamos y_i a cada valor representativo del zscore de x_i de la serie de todos los "i"ésimos valores de X, tenemos:

$$y_i = \frac{x_i - \mu}{\sigma}$$

Visto en código Python sería algo así:

```
data['OBV_acotado'] = (data.OBV - data.OBV.rolling(n).mean()) / (data.OBV.rolling(n).std())
data['OBV_acotado'].iloc[-500:].plot(figsize=(15,5), title=f'TSLA OBV_{n} acotado por zscore')
```



Ya no queda acotado entre 0 y 1, sino que la cota dependerá de que tanto outlier tenga la distribución, es decir de que tan altos sean los zscores en cantidad de desvíos de la media en los puntos más extremos. Esto no necesariamente es malo, en definitiva este valor es mas "expresivo" ya que tiene un significado mas concreto que el oscilador entre 0 y 1. También los hicimos dentro de una ventana de "n" velas, en el próximo ejemplo lo mostramos con el zscore representativo de toda la distribución.

La 3º Forma es por zscores, pero en lugar de calcular el μ y σ por la ventana de "n" velas, los tomamos de toda la distribución

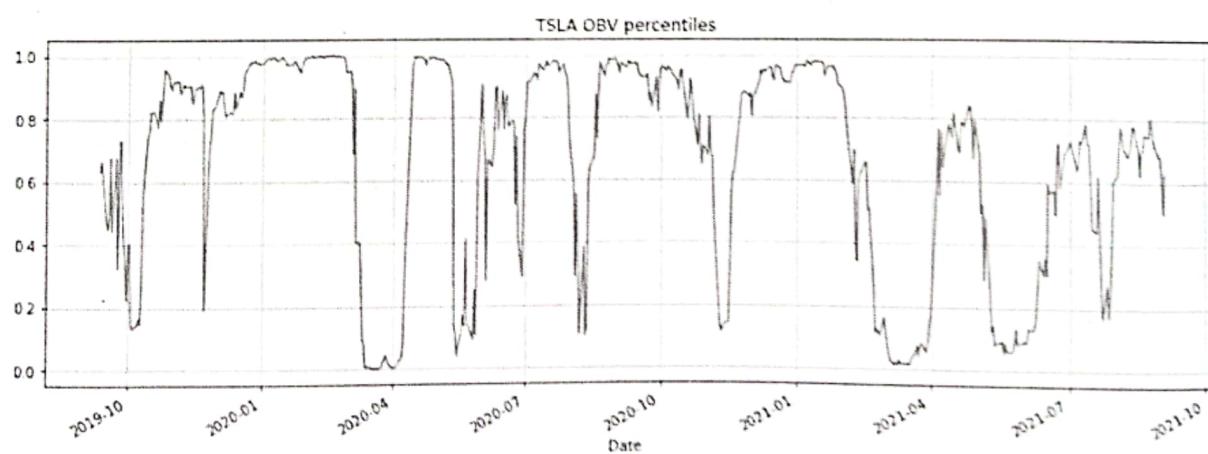
```
data['OBV_acotado'] = (data.OBV - data.OBV.mean()) / (data.OBV.std())
df = data['OBV_acotado'].iloc[-500:]
df.plot(figsize=(15,5), title=f'TSLA OBV_{n} acotado por zscore', grid=True)
```



Fíjense la gran diferencia, si bien, en ambos casos estoy acotando el oscilador en un rango de +/- la cantidad de sigmas fuera de la media, en realidad esta segunda opción guarda la memoria de toda la distribución, es decir me marca mejor los efectos estacionales que hacerlo con una ventana acotada. Son diferencias, no es que haya uno mejor que otro, todo es cuestión de ver en un backtest cual da mejor resultado en cada tipo de estrategia a operar. Sin embargo, claramente cumple mejor las características de un oscilador el caso anterior tomando una ventana acotada.

La 4º forma de desestacionalizar y acotar un indicador es la mas simple de todas, simplemente calcular el percentil de cada valor en su distribución, el percentil es el quantil n=100 en formato decimal

```
titulo = "TSLA OBV percentiles"
data.OBV.rank(pct=True).iloc[-500:].plot(figsize=(15,5), grid=True, title=titulo)
```



Así que bueno, ahí tienen 4 formas diferentes de convertir un valor que no tiene una banda sobre la que se mueve a un indicador acotado dentro de una banda fija, o sea un oscilador. Así que desde ahora asumo que esto queda en el background de herramientas, yo para simplificar probablemente no armé indicadores muy rebuscados, pero está bueno ya saber que cuando tengan un indicador muy estacional o que no se pueda usar para backtest por su gran variación según la época, pueden convertirlo en oscilador para poder usarlo como indicador a través de cualquier rango de fechas y poder usarlo en backtesting de estrategias.

Retomamos Indicadores de flujos (volumen y derivados)

Retomamos, entonces con lo que veníamos, vemos ahora para cerrar esta categoría un indicador de flujo basado en derivados, en este caso voy a usar la tasa de fondeo de los futuros del BTC, es un indicador sobre el que se ha hablado mucho en este mercado, tiene sus cosas, no es tan sencillo encontrarle la vuelta para meterlo en una estrategia de trading, pero desde ya que es información más que útil.

De paso repasemos la API de deribit que es un exchanges de futuros y opciones de BTC y ETH por el momento, es el de mayor liquidez del mundo crypto en opciones y está hace muchos años con lo cual es de gran confianza y tiene una buena base de datos históricos totalmente gratuitos para usar.

Antes que nada, tengo que armarme rangos de fechas en formato timestamp (milisegundos)
Y estos rangos deben abarcar mas o menos un mes cada rango

Esto se debe a que la API de deribit me devuelve máximo de históricos de a 1000 filas, y como las tasas de funding son cada 1 hora (mercado 24x7) en un mes entran 744 como máximo, así que haremos una consulta cada un mes

```
import time

periodos = 10
ts = [int(time.time())*1000 - 3600*1000*24*30*x for x in range(periodos,-1,-1)]
rangos = [(ts[i], ts[i+1]) for i in range(len(ts)-1)]
```

```
[(1605209779000, 1607801779000),
(1607801779000, 1610393779000),
(1610393779000, 1612985779000),
(1612985779000, 1615577779000),
(1615577779000, 1618169779000),
(1618169779000, 1620761779000),
(1620761779000, 1623353779000),
(1623353779000, 1625945779000),
(1625945779000, 1628537779000),
(1628537779000, 1631129779000)]
```

O sea, tomé desde el momento presente, 10 meses para atrás (10 períodos de 30 días)

El *1000 es porque la API necesita el timestamp en formato milisegundos

Usaré estos rangos para hacer cada query a la API

Armamos el script para hacer las 10 queries a la API

```
import pandas as pd, requests, time

data = pd.DataFrame()
for i, rango in enumerate(rangos):
    print(f"Procesando ts {rango} N° {i+1}/{len(rangos)}", end='\r')
    url = 'https://deribit.com/api/v2/public/get_funding_rate_history'
    params = {"end_timestamp": rango[0], "instrument_name": "BTC-PERPETUAL", "start_timestamp": rango[1]}
    df = pd.DataFrame(requests.get(url, params).json()['result'])
    df.set_index('timestamp', inplace=True)
    df.index = pd.to_datetime(df.index, unit='ms')
    data = pd.concat([data, df])
    print(f"Procesando ts {rango} N° {i+1}/{len(rangos)}", end='\r')

Procesando ts (1628537779000, 1631129779000) N° 10/10
```

Y graficamos nuestro indicador para verificar se tiene sentido

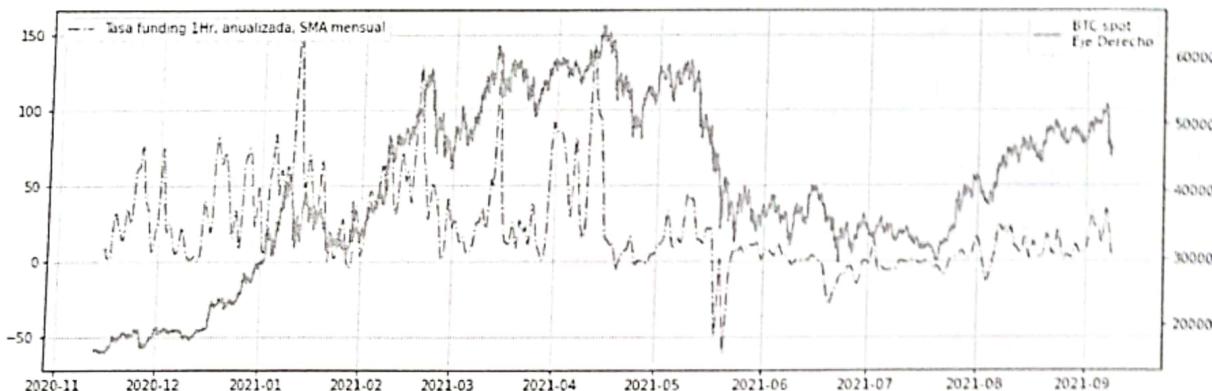
A todo esto, anualizo la tasa para que sea mas amigable conceptualmente la lectura, y le hago una media móvil simple a esa tasa solo para que el gráfico quede mas limpio, esto no es la mejor idea para usar como indicador porque suaviza el efecto temporal, pero para mostrarlo y visualizarlo uno está bien hacerlo.

```
import matplotlib.pyplot as plt

data['tasa'] = data.interest_1h.rolling(24).mean().mul(24).mul(100).mul(365)

fig, ax = plt.subplots(figsize=(15,5))
ax.plot(data.tasa.rolling(30).mean(), label='Tasa funding 1Hr, anualizada, SMA mensual')
ax.legend(loc='upper left')
ax.grid()

ax2 = ax.twinx()
ax2.plot(data.index_price, c='red', label='BTC spot\nEje Derecho')
ax2.legend(loc='upper right')
plt.show()
```



Bueno, ya saben como transformar este indicador en oscilador dentro de un rango si quisieran así que les dejo la idea picando. Y dejo esta categoría de indicadores acá porque si no este capítulo de indicadores se va a hacer eterno y la idea es concentrarse después en el backtesting en sí.

Lo interesante de llevar los indicadores al formato de oscilador es que empiezan a representar mas las tasas de cambio o las tendencias del mismo indicador, en lugar de su valor absoluto, con lo cual por lo general terminan aumentando la sensibilidad del mismo.

Combinados, Construcción de indicadores propios:

Muchas veces nos pasa que tenemos un indicador, pero notamos que dicho indicador es muy sensible a otro dato, el ejemplo que voy a poner a continuación es arbitrario, en realidad para reutilizar código anterior y para usar series de datos que ya tenemos todo en una serie y no andar bajando data de diferentes fuentes, pero la idea de esta explicación que viene a continuación es que lo usen para construir sus indicadores combinados de modo general, con lo que se les ocurra.

Dicho esto, supongamos que asumimos que el RSI es sensible al volumen y queremos construir un indicador personalizado que sea un oscilador tal cual como el RSI pero "ponderado" por saltos de volumen o algo así.

Podemos armar un RSI relativo primero, esto no es trivial, el RSI al ser un oscilador tiene banda superior e inferior, pero quizás signifiquen lo mismo en términos de intensidad para usarlo como indicador combinado.

Es decir por ejemplo que un RSI de 10 tiene la misma potencia que uno de 90, porque ambos están a 40 de distancia de su media que es 50, asimilando que la media dará 50, podría afinarse más aun el cálculo pero para no complicarla asumamos eso.

Entonces el RSI común ahora lo transformamos es uno que siga la fórmula:

$$\hat{RSI} = |RSI - \overline{RSI}|$$

Esto me produce ya de por sí un indicador más expresivo de los valores extremos que del momentum, veámoslo:

```
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt

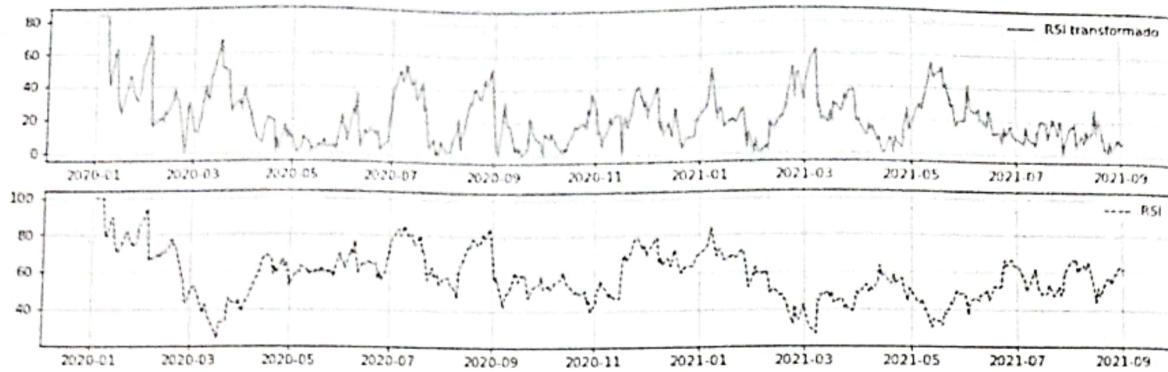
data = yf.download('TSLA', start='2020-01-01', end='2021-09-04')

ruedas = 14
data['dif'] = data['Close'].diff()
data['win'] = np.where(data['dif'] > 0, data['dif'], 0)
data['loss'] = np.where(data['dif'] < 0, abs(data['dif']), 0)
data['ema_win'] = data.win.ewm(alpha=1/ruedas).mean()
data['ema_loss'] = data.loss.ewm(alpha=1/ruedas).mean()
data['rs'] = data.ema_win / data.ema_loss
data['rsi'] = 100 - (100 / (1+data.rs))

data['rsi_abs'] = (data.rsi - data.rsi.mean()).abs()**2

fig, ax = plt.subplots(figsize=(15,5), nrows=2)
ax[0].plot(data.rsi_abs, color='gray', label='RSI transformado')
ax[0].legend(loc='upper right')
ax[0].grid()

ax[1].plot(data.rsi, c='k', lw=1, ls='--', label='RSI')
ax[1].legend(loc='upper right')
ax[1].grid()
plt.show()
```



A veces se suele multiplicar x2 a este tipo de transformación simétrica de respecto de la media para que el valor transformado mantenga la misma escala que el original, es un detalle de expresión del indicador transformado ya que multiplicar por una constante no afecta a la correlación que vaya a tener con cualquier forward o con otro activo porque es una transformación lineal.

Bueno, ahora que ya tenemos el transformado, podemos calcular un oscilador de volumen para ajustar este RSI transformado ponderándolo por el oscilador de volumen, veamos:

```

import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt

data = yf.download('TSLA', start='2020-01-01', end='2021-09-04')

# Calculo de RSI comun
ruedas = 14
data['dif'] = data['Close'].diff()
data['win'] = np.where(data['dif'] > 0, data['dif'], 0)
data['loss'] = np.where(data['dif'] < 0, abs(data['dif']), 0)
data['ema_win'] = data.win.ewm(alpha=1/ruedas).mean()
data['ema_loss'] = data.loss.ewm(alpha=1/ruedas).mean()
data['rs'] = data.ema_win / data.ema_loss
data['rsi'] = 100 - (100 / (1+data.rs))

# RSI TRansformado
data['rsi_abs'] = (data.rsi - data.rsi.mean()).abs()*2

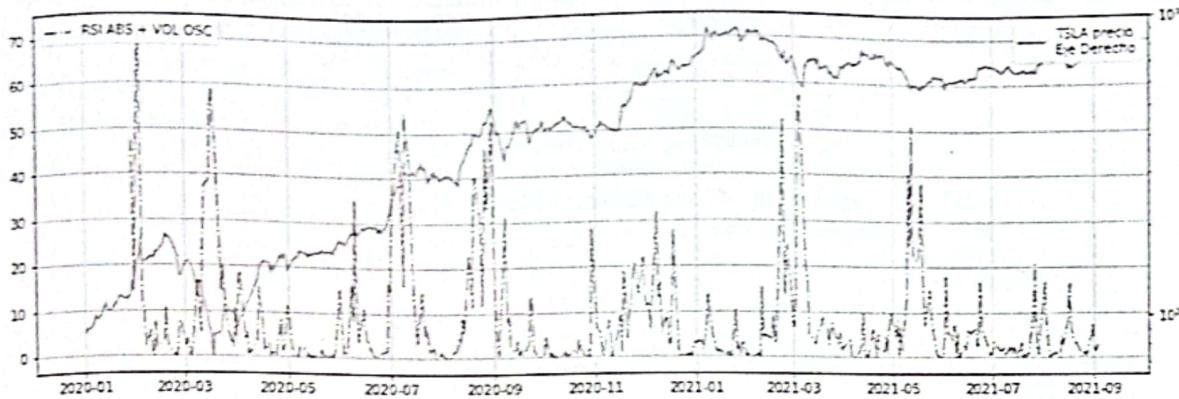
# Oscilador de Volumen
v_min = data.Volume.rolling(20).min()
v_max = data.Volume.rolling(20).max()
data['vol_osc'] = (data.Volume - v_min) / (v_max - v_min)

# RSI transformado ajustado por volumen
data['RSI_VOL'] = data.vol_osc * data.rsi_abs

# Gráfico
fig, ax = plt.subplots(figsize=(15,5))
ax.plot(data.RSI_VOL, ls='-.', label='RSI ABS + VOL OSC')
ax.legend(loc='upper left')
ax.grid()

ax2 = ax.twinx()
ax2.plot(data.Close, c='red', label='TSLA precio\nEje Derecho')
ax2.legend(loc='upper right')
ax2.set_yscale('log')
plt.show()

```



Y listo, me queda armado el indicador de valores extremos de RSI ajustados por volumen, indicador que me marcará en sus picos excesiva euforia o miedo con gran volumen, podría ser un buen indicador para una estrategia contrarian.

Les dejo una idea, en unas páginas cuando veamos indicadores estadísticos y paramétricos de las distribuciones, bueno, estos parámetros son ideales para ajustar indicadores, por ejemplo indicador de volumen ajustado por volatilidad (sigma) de precio, o indicadores de cruces de medias (momentum) ajustados por volatilidad de volumen (flujo), en fin las cosas que se pueden armar son infinitas casi teniendo las herramientas.

Conteos discretos

Este tipo de herramienta suma mucho, en realidad son buenos como clasificadores, ya que la discretización tiende mas a escenarios binarios o con pocos estados posibles, ejemplo, supongamos que en lugar de usar el % que sube o baja el precio un día, usamos el color de la vela, o contamos los días que sube respecto de la cantidad de días que baja en una ventana, sin importar cuento sube o baje

Por poner un ejemplo sencillo, vamos a generar al mismo tiempo 3 indicadores discretos parecidos, vamos a contar para las últimas 100 velas lo siguiente:

- La cantidad de velas de días alcistas: cuyo cierre es mayor al de la vela anterior
- La cantidad de velas con gap alcista: cuya apertura es mayor al cierre anterior
- La cantidad de velas verdes: cuyo cierre es mayor a la apertura de la misma vela

Repite, no nos interesa la intensidad, da lo mismo si un día sube 1% o 2%, en este caso contabilizamos cantidades, por eso es un conteo discreto, y luego las sumamos dentro de una ventana, como dijimos 100 velas hacia atrás en cada punto

Veamos como queda esto en código y grafiquemos las series y un KDE de sus distribuciones.

```

import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt

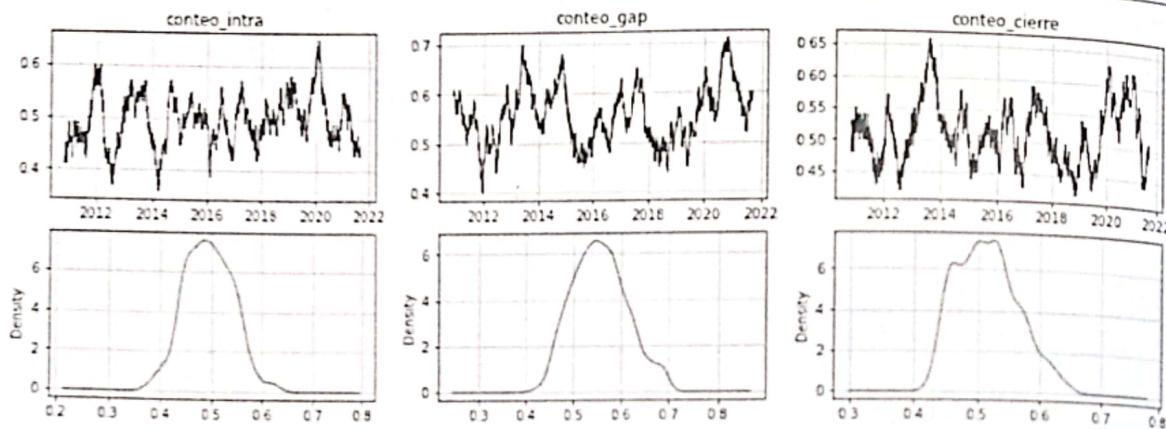
data = yf.download('TSLA', auto_adjust=True, end='2021-09-04')
data['tipo_cierre'] = np.where(data.Close > data.Close.shift(), 1, 0)
data['tipo_gap'] = np.where(data.Open > data.Close.shift(), 1, 0)
data['tipo_intra'] = np.where(data.Close > data.Open, 1, 0)

n = 100
data['conteo_cierre'] = data['tipo_cierre'].rolling(n).sum() / n
data['conteo_gap'] = data['tipo_gap'].rolling(n).sum() / n
data['conteo_intra'] = data['tipo_intra'].rolling(n).sum() / n

fig, ax = plt.subplots(figsize=(15,5), ncols=3, nrows=2)
for i in range(3):
    ax[0,i].plot(data.iloc[:, -i-1], lw=1, c='k')
    ax[0,i].grid()
    ax[0,i].set_title(data.columns[-i-1])

    data.iloc[:, -i-1].plot.kde(ax=ax[1,i], grid=True)

```



Como ven, si bien usé 100 velas, en el código dividido por "n" para que, use la ventana que use, el indicador me da una especie de porcentaje de velas positivas en intradiario, gap y diario

Es esperable que las medias den en torno a 0.5 es decir la mitad de los días para cada lado, es mas, como el 0% de cambio no lo contamos porque en las condiciones usamos un ">" y no un ">=", es esperable que las medias den en torno a 0.47 o por ahí.. sin embargo, vemos que concretamente la de los gaps da bastante más alta.

Nada, lo dejo ahí como dato curioso, después la idea es que investiguen mejor cada vez que crean un indicador a que se deben esas "anomalías".

Otra cosa interesante que se ve en el indicador de los conteos de cierre[n]/cierre[n-1] es que la distribución es asimétrica, es decir, la mitad positiva está más "estirada" por decirlo en bruto, bueno, lo mismo, un buen trabajo de data mining sería tratar de "hacer hablar" a este indicador en el sentido de desmenuzar los datos, desagregarlos, clasificarlos y estudiarlos mucho mejor para poder usarlos luego con mas éxito en la toma de decisiones o armado de la estrategia, yo acá simplemente les muestro como construir distintos tipos de indicadores.

Indicadores de Estacionalidad

Este tipo es muy interesante, puede ser usado de muchas formas diferentes, en cuanto a backtesting, si solo nos vamos a basar en series de precios, lo que se puede verificar es si no existe algún tipo de estacionalidad en las series ya sea:

- Por día de la semana
- Por horario del día
- Por semana del año
- Por día del mes

Hay miles de factores que pueden influir en que se marque algún tipo de sesgo en las series en alguno de esos segmentos que les puse, el mas de sentido común quizá sea el mas largo, la semana del año, esto puede tener que ver con muchas cosas, desde factores climáticos, festivos o culturales, en fin, dependiendo los instrumentos y el mercado que operen puede que haya algún tipo de estacionalidad en ciertas series que puede servirnos como indicador.

En los timeframes mas chicos puede tener que ver con factores de liquidez ya que los horarios mas movidos los precios pueden comportarse ligeramente diferente a los horarios con menor movimiento de flujos, obviamente mención aparte los horarios cercanos a la apertura y el cierre son muy particulares, con respecto al día del mes y día de la semana vamos un toque al mundo crypto

Antes de mostrarles el siguiente ejemplo deben instalarse este paquete (que escribí yo) que tiene algunas herramientas para el mundo crypto en realidad mas para defi, todas gratuitas sin login ni token ni nada, en este caso lo vamos a usar para traer precios de coingecko que tiene miles de cryptomonedas y sus datos históricos entre otras cosas, les super recomiendo esta librería para ver y escanear muchos activos que cotizan en diferentes exchanges, porque las APIs de cada Exchange solo tiene las cryptomonedas que tienen listadas, pero coingecko tiene todas.

Bueno, miren este código:

```
# pip install defi

import defi.defi_tools as dft
import numpy as np

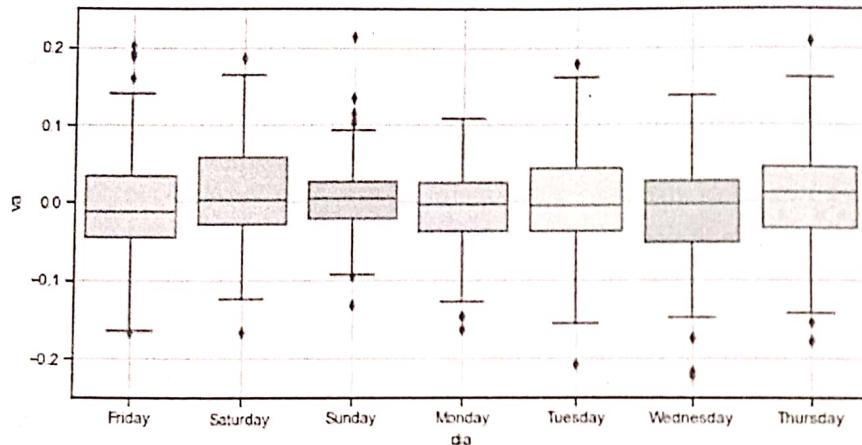
df = dft.geckoHistorical('algorand')
df['va'] = np.log(df.price / df.price.shift())
agg = df.groupby(df.index.day_name()).va.mean()
agg.sort_values()
```

```
date
Wednesday -0.013204
Friday      -0.007116
Monday      -0.006723
Tuesday     -0.001721
Sunday       0.002129
Thursday    0.008336
Saturday    0.020628
Name: va, dtype: float64
```

Como ven el código es super sencillo, se baja los datos de "algorand" en una línea y luego calcula en la siguiente los retornos logarítmicos, y finalmente arma un agregado de rendimientos por día de la semana, y como ven... terribles diferencias entre días de semana, lo van a ver mejor en un grafo de boxplots:

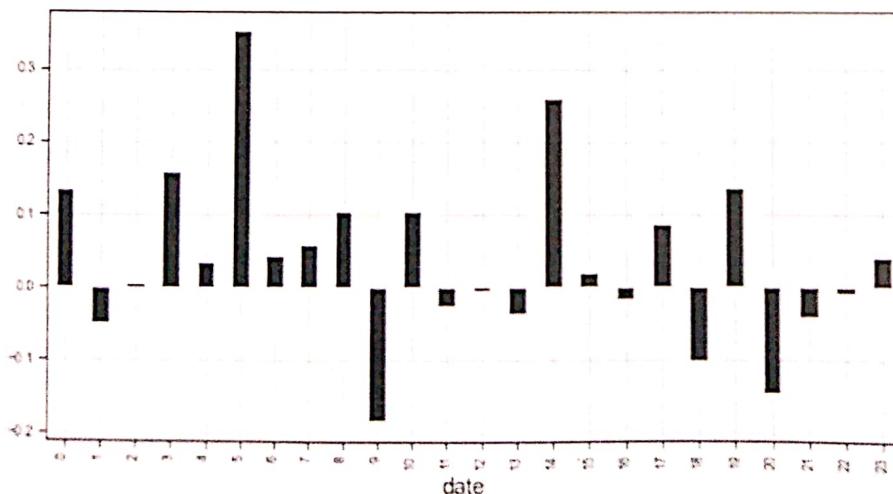
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,5))
df['dia'] = df.index.day_name()
sns.boxplot(x="dia", y="va", data=df)
plt.ylim(-0.25, 0.25)
plt.grid()
```



Estas diferencias en el mundo crypto son muy comunes, tienen infinidad de explicaciones, en algunas casos hasta son burdas, por ejemplo hay programas de staking que entregan las recompensas en forma semanal siempre el mismo día, con lo cual están aumentando instantáneamente la oferta y de forma sistemática siempre el mismo día de la semana, en fin, hay muchas explicaciones lo importante para nosotros es poder detectar las anomalías y poder luego encontrar métodos eficientes de operarlas. Miren el siguiente gráfico, la misma cryptomoneda por hora del día.

```
df = dft.geckoHistorical('algorand', days=90)
df['va'] = np.log(df.price / df.price.shift())*100
agg = df.groupby(df.index.hour).va.mean()
agg.sort_index().plot(kind='bar', color='k', figsize=(10,5), grid=True)
```



Como les decía los motivos pueden ser muy pero muy diversos, el punto es que si detectamos que sistemáticamente en un determinado horario y/o día determinada moneda tiene mucho mejor performance y otra moneda tiene peor performance, esto se presta para arbitrajes estadísticos, que a la larga neutralizando riesgo sistémico del ecosistema crypto (o incluso buscando pares con mayor correlación que lo típico) puede ser muy bien aprovechado.

Indicadores Estadísticos o paramétricos

Este tipo de indicadores es en mi opinión de los mas ricos, mas que nada porque son super descriptivos y más fáciles de interpretar, por lo general son parámetros inherentes a la distribución de los datos, por eso algunos los llaman "indicadores paramétricos" también

Les dejo una lista de los que yo considero mas útiles y comunes, obviamente que sobre ellos se pueden hacer todo tipo de construcción, ajuste por otros indicadores, transformación en osciladores, etc.

Ejemplo, si usamos como indicador un `rolling(n).std()` representando a la volatilidad de la serie, bien podríamos hacer un `rolling(q).std()` con $q > n$ representando una ventana mas amplia y por tanto con reacción mas lenta a cambios en la volatilidad, luego podríamos armar un cruce de medias de volatilidades, digo por decir algo simple, pero así como esto, pueden armarse todo tipo de indicadores basados en este tipo de parámetros.

Los más conocidos son la varianza, skew y kurtosis, que son el 2do, 3er y 4to momento centrado del mu respecto al desvío que se usa para ajustar al parámetro en términos relativos elevándolo a la potencia del momento del mu en cuestión. Son muy útiles porque dada su construcción, los tres describen fenómenos bien diferentes, la varianza al ser el segundo momento, eleva al cuadrado la diferencia entre X_i y la media μ , esto hace que sea agnóstico al signo de los retornos, ya que cualquier cosa negativa elevada al cuadrado da lo mismo que si elevara el mismo valor positivo. Este fenómeno ya no pasa si se eleva al cubo la misma diferencia (skew) por lo que este indicador elevando al cubo me dará una medida de para que lado del 0 esta mas densa la distribución, y por último elevando a la 4ta potencia, lo que pasa es que por un lado las colas pesadas, es decir valores muy positivo o muy negativos hacen que el $(X_i - \mu)^4$ sea muy alto pero como la kurtosis normaliza dividiendo por el σ^4 si hay muchos valores muy cercanos a 0 hacen que el denominador sea muy alto, elevando la kurtosis también, con lo que se observan kurtosis muy altas tanto por colas pesadas como por centros muy "puntiagudos", en fin es todo un tema este de los indicadores paramétricos, en el t[6] desarrollé un poco mas el tema por si les interesa.

Fórmulas de varianza, skewness y kurtosis

$$\sigma_n^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2 \quad skew = \sum_{i=1}^n \frac{(X_i - \bar{X})^3}{\sigma^3} \quad kurtosis = \sum_{i=1}^n \frac{(X_i - \bar{X})^4}{\sigma^4}$$

Bueno, también hay otros indicadores como ratios de colas como el de Rachev y también construcciones con quantiles y zscores, y por último parametrización no-normal, es decir asumiendo otro tipo de distribución de variable continua, nuevamente repasar t[6] de esta colección donde este tipo de cosas está mucho mejor detallada, simplemente les armo una listita de este tipo de indicadores:

- Indicadores paramétricos de la distribución:
 - Desvio estándar
 - skew
 - kurtosis
 - mu
 - tails ratios
- Zscores de cualquier indicador de las series anteriores
- Fiteo de parámetros de otras dist

Y vemos entonces la implementación en Python de algunos de estos:

```
import yfinance as yf
import pandas as pd

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04')
data['retornos'] = data['Close']/data['Close'].shift() - 1

roll = data.retornos.rolling(40)
data['sigma_40'] = roll.std() * 250**0.5
data['kurtosis'] = roll.apply(pd.DataFrame.kurtosis)
data['tails_ratio'] = roll.quantile(.95) / roll.quantile(.05).abs()
data.dropna()
```

Date	Open	High	Low	Close	Volume	retornos	sigma_40	kurtosis	tails_ratio
2010-08-25	3.832000	3.996000	3.712000	3.980000	2516500	0.036458	0.837087	1.406638	0.809310
2010-08-26	3.978000	4.054000	3.920000	3.950000	2169000	-0.007538	0.837157	1.400599	0.809310
2010-08-27	3.950000	3.974000	3.900000	3.940000	1898000	-0.002532	0.814656	1.904363	0.859729
2010-08-30	3.940000	4.038000	3.922000	3.974000	3064000	0.008629	0.749911	2.679014	1.088771
2010-08-31	3.932000	3.958000	3.886000	3.896000	1005500	-0.019628	0.826114	0.124591	1.396087
...									
2021-08-30	714.719971	731.000000	712.729980	730.909973	18804200	0.026674	0.342351	-0.484144	1.351535
2021-08-31	733.000000	740.390015	726.440002	735.719971	20855400	0.006581	0.333409	-0.347752	1.525959
2021-09-01	734.080017	741.989990	731.270020	734.090027	13204300	-0.002215	0.327214	-0.210943	1.525959
2021-09-02	734.500000	740.969971	730.539978	732.390015	12777300	-0.002316	0.326658	-0.178128	1.525959
2021-09-03	732.250000	734.000000	724.200012	733.570007	15246100	0.001611	0.326569	-0.169752	1.525959

2777 rows × 9 columns

Como ven armo una ventana Rolling() de 40 velas, y sobre ella construyo diferentes rollings, sigma, kurtosis y ratio de Rachev (el cálculo de Rachev así nomás solo con los quantiles no es la mejor idea, mas adelante cuando veamos ratios de riesgo en el backtest lo desarollo mejor, por ahora simplemente para presentar el tema lo mando así, pero la idea va a ser tomar toda el área bajo las colas en lugar de los valores de quantiles .05 y .95)

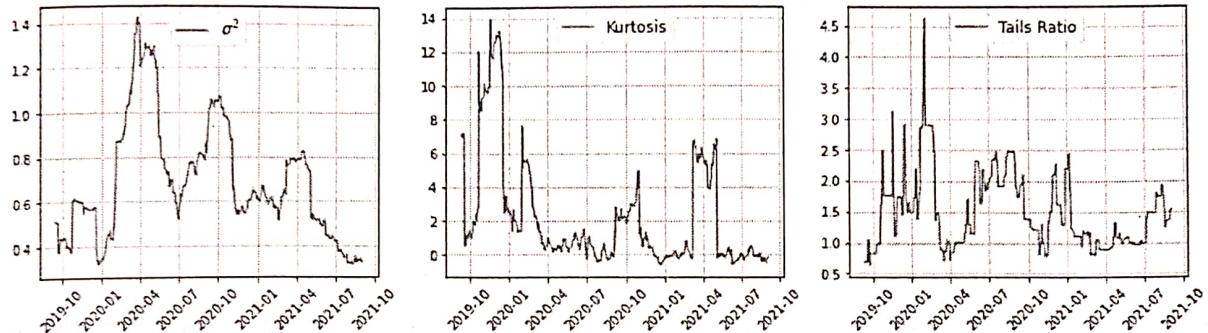
Veamos estos indicadores en gráficos ya que estamos:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(16,4), ncols=3)
df = data.iloc[-500:]

ax[0].plot(df['sigma_40'], label=r'$\sigma^2$')
ax[1].plot(df['kurtosis'], label='Kurtosis')
ax[2].plot(df['tails_ratio'], label='Tails Ratio')

for i in range(3):
    ax[i].grid()
    ax[i].legend(loc='upper center', fontsize=12)
    ax[i].tick_params(axis='x', rotation=45)
```



Como hablaba antes, a partir de estos se pueden armar osciladores, cruces fast/slow etc, yo personalmente uso mucho los indicadores de cambio de régimen de volatilidad para el trading de opciones, y son indicadores muy backtesteadables.

Referenciales, benchmarks

Bueno, llegamos a los indicadores mas de contexto, estos sirven para ampliar el campo de visión ya que muchas veces se hacen backtests y estrategias de trading algorítmico mirando solo el activo a operar y sin tener en cuenta el entorno del mercado en el que están pasando las cosas. Hacer eso sería muy pobre ya que por ejemplo no es lo mismo considerar una ruptura de un indicador fuerte en un contexto normal (ruptura por algo endógeno al instrumento) que analizar la misma ruptura pero en un contexto en el que el mercado de referencia se hunde o tiene una explosión alcista.

Para mí una estrategia de trading algorítmico seria debe tener al menos 2 o 3 indicadores referenciales como mínimo, ya que si no es analizar el instrumento totalmente a ciegas del contexto lo cual es, como decía demasiado pobre. Obviamente que en los ejemplos que vamos a ver para ir calculando ratios y demás voy a simplificar, y es por eso que armo este listado de mi top10 de indicadores de trading, imagínense que si voy a armar u ejemplo con 3 indicadores de cada tipo es un código que ya de por si tiene más de una página entera para mostrar solamente la construcción de los features o inputs del modelo. Perdón que repita mucho lo mismo, pero es importante entender que un backtest profesional son al menos 1000 líneas de código y que los ejemplos en un libro no deben sobrepasar las 20 o 30 líneas para que sea didáctico.

Les dejo una lista de indicadores referenciales, no son todos obviamente, pero si los más típicos:

- Ratios de correlación:
 - Alfa
 - Beta
 - Sharpe, Sortino, Jensen α
 - Information Ratio
 - Cov, rho, entre subyacentes
 - Cov, rho, entre indicadores
- Mercado de referencia ej Merval, SP500 etc, ETFs de sectores representativos de la industria
- Commodities de referencia que incidan mucho en la industria
- Monedas FX (muy útiles en la banca y en economías muy inestables financieramente)
- Indicadores macro de economía (empleo, actividad, sectoriales, IPC, etc)
- Otros subyacentes similares (misma industria, mismo clúster... veremos más adelante en algoritmos de clusterización)

Como es usual veremos la implementación en Python de algunos de ellos para bajarlo un poco a tierra, la idea en general es como les anticipaba, ver en qué CONTEXTO pasan las cosas, pero antes veamos la formulación de los 4 indicadores que voy a calcular:

Asumiendo el vector R de los retornos del activo y R_b del benchmark (en nuestro ejemplo usaremos a TSLA como activo y al SP500 como benchmark), tenemos las siguientes expresiones para alfa y beta en la serie para cada punto:

$$\alpha = R - R_b \quad \beta = \frac{Cov(R - R_b)}{\sigma_{R_b}^2}$$

$$\overline{\alpha_w} = \sum_{i=1}^w \frac{R_i - R_{b_i}}{w} \quad \overline{\beta_w} = \frac{\sum_{j=1}^w (R_j - \overline{R})(R_{b_j} - \overline{R_b})}{\sum_{j=1}^w (R_{b_j} - \overline{R})^2}$$

Lo mismo con el ratio sharpe y el ratio de información:

$$sharpe = \frac{R - F_{RISK}}{\sigma_R}$$

$$\overline{sharpe}_w = \frac{\sum_{i=1}^w R_i - F_{risk}}{\frac{1}{w} \sum_{i=1}^w \sum_{j=1}^n \frac{(R_i - \overline{R})^2}{n}}$$

$$IR = \frac{R - R_b}{\sigma_{(R-R_b)}} = \frac{\alpha}{\sigma_\alpha}$$

$$\overline{IR}_w = \frac{\sum_{i=1}^w \frac{R_i - R_{b_i}}{w}}{\sum_{i=1}^w \frac{(R_i - R_{b_i} - \overline{R} - \overline{R_b})^2}{w}}$$

Antes de avanzar con la implementación veamos conceptualmente los significados:

- Alpha: Exceso de retorno del activo respecto al mercado (mayor a cero es que el activo rinde más que el mercado y menor a cero que el activo rindió por debajo que el mercado)
- Beta: Exceso de volatilidad respecto al mercado. Es decir por cada 1% que varía el mercado cuanto varía en promedio el activo, mayor a 1 significa que el activo es mas volátil que el mercado y menor a 1 lo contrario
- Sharpe: Es el retorno ajustado por volatilidad, es decir por cada 1% de volatilidad cuanto es el retorno
- Ratio de Información: Es el retorno puro del activo (limpio del retorno del mercado) ajustado por la volatilidad neta (volatilidad del retorno puro, limpio del efecto mercado).

Recomiendo que si no terminaron de entender conceptualmente los 4 indicadores, repasen la lectura y/o amplíen en internet porque realmente son fundamentalísimos los 4. Ahora sí vamos entonces a la implementación de esto en Python:

```
import yfinance as yf
import numpy as np

data = yf.download(['TSLA','SPY'], auto_adjust=True, end='2021-09-04')['Close']
data['R'] = data['TSLA']/data['TSLA'].shift() -1
data['Rb'] = data['SPY']/data['SPY'].shift() -1
data['alpha'] = data.R - data.Rb
data['sigma_40'] = data.R.rolling(40).std()

rf = 0
w = 100
roll = data.rolling(w)
data['rolling_alpha'] = roll.alpha.mean()
data['rolling_beta'] = roll.R.cov(data.Rb) / roll.Rb.var()

data['rolling_sharpe'] = roll.R.mean() / roll.sigma_40.mean()
data['rolling_IR'] = roll.alpha.mean() / roll.alpha.std()
data.dropna()
```

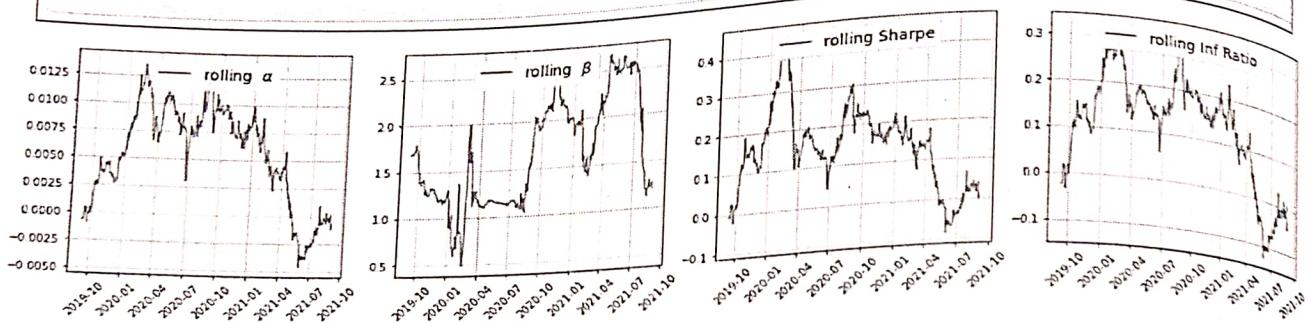
Date	SPY	TSLA	R	Rb	alpha	sigma_40	rolling_alpha	rolling_beta	rolling_sharpe	rolling_IR
2011-01-14	105.272865	5.150000	-0.017925	0.007245	-0.025170	0.042133	0.001593	0.975477	0.095350	0.039457
2011-01-18	105.451965	5.128000	-0.004272	0.001701	-0.005973	0.042053	0.001208	0.967241	0.085266	0.029997
2011-01-19	104.417969	4.806000	-0.062792	-0.009805	-0.052987	0.042585	0.000686	1.068941	0.071416	0.016893
2011-01-20	104.279549	4.524000	-0.052677	-0.001326	-0.057351	0.041192	0.000293	1.144447	0.057264	0.007148
2011-01-21	104.515648	4.608000	0.018568	0.002264	0.016303	0.040823	0.000225	1.217706	0.059904	0.005487
...
2021-08-30	452.230011	730.909973	0.026674	0.004398	0.022277	0.021652	-0.000072	1.239499	0.032777	-0.002906
2021-08-31	451.559998	735.719971	0.006581	-0.001482	0.008062	0.021087	0.000181	1.261682	0.038547	0.007321
2021-09-01	451.799988	734.090027	-0.002215	0.000531	-0.002747	0.020695	-0.000212	1.267250	0.025860	-0.008683
2021-09-02	453.190002	732.390015	-0.002316	0.003077	-0.005392	0.020660	-0.001096	1.226436	-0.003684	-0.047824
2021-09-03	453.079987	733.570007	0.001611	-0.000243	0.001854	0.020654	-0.000717	1.190308	0.010303	-0.031654

2678 rows x 10 columns

Graficamos para los últimos 2 años:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(20,4), ncols=4)
df = data.iloc[-500:]
ax[0].plot(df['rolling_alfa'], label=r'rolling $\alpha$')
ax[1].plot(df['rolling_beta'], label=r'rolling $\beta$')
ax[2].plot(df['rolling_sharpe'], label='rolling Sharpe')
ax[3].plot(df['rolling_IR'], label='rolling Inf Ratio')

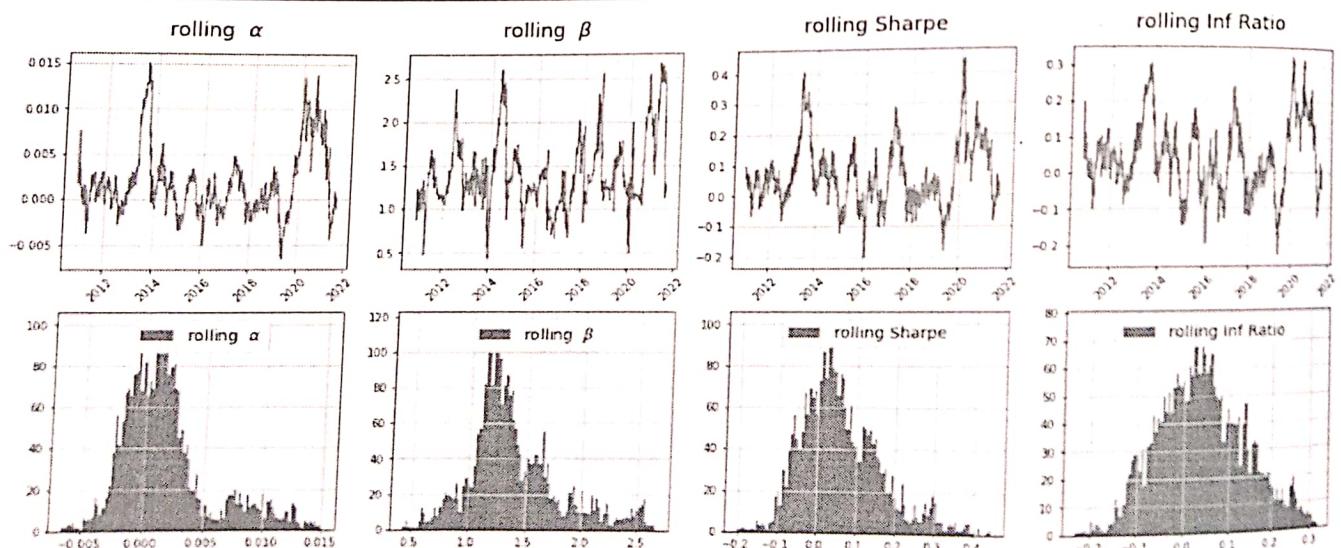
for i in range(4):
    ax[i].grid()
    ax[i].legend(loc='upper center', fontsize=12)
    ax[i].tick_params(axis='x', rotation=45)
```



O emprójámos un poco el código y lo vemos con histograma para los últimos 10 años, como ven todos estos indicadores bastante acotados oscilan naturalmente:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(20,8), ncols=4, nrows=2)
series = {'rolling_alfa': r'rolling $\alpha$', 'rolling_beta': r'rolling $\beta$', 'rolling_sharpe': 'rolling Sharpe', 'rolling_IR': 'rolling Inf Ratio'}
for i, item in enumerate(series.items()):
    ax[0][i].plot(data[item[0]], label=item[1])
    ax[0][i].grid()
    ax[0][i].legend(loc='upper center', fontsize=14)
    ax[0][i].tick_params(axis='x', rotation=45)

    ax[1][i].hist(data[item[0]].dropna(), bins=100)
    ax[1][i].grid()
```



Indicadores de sentimiento

La idea de este tipo de indicadores es buscar un sentimiento social de un subyacente en redes sociales con un claro foco en la inmediatez. Es decir se trata de barrer automáticamente muchos posteos en redes sociales y muy pocos segundos y ver por ejemplo que reacción masiva hay en los comentarios a un evento por ejemplo "un discurso esperado de la FED donde posiblemente defina política de tasas", también puede hacerse el rastreo sin evento alguno, simplemente rastrear el sentimiento en las redes.

Este tipo de indicadores está muy de moda pero tiene dos aristas importantes a considerar por las cuales muchos sistemas que vi últimamente basados en este tipo de indicador fallan:

- Alcance real del muestreo vs Población
- Herramienta de NLP para interpretar el sentimiento del posteo

Respecto al alcance, la realidad es que las APIs de redes sociales tienen ciertos límites de requests. Ejemplo, twitter: que varían entre 15 y 1500 requests cada 15 minutos, es bastante complejo el esquema de los requests, pueden chequear mas detalles del alcance de la API gratuita en: <https://developer.twitter.com/en/docs/twitter-api/v1/rate-limits>

Ahora si queremos hacer un buen sistema de sentimiento, deberíamos barrer cientos de miles, y si se puede millones de datos. Para grandes volúmenes de datos hay que partir bien arriba de los USD 3000 por mes y ciertamente no está al alcance de un trader independiente y mucho menos para hacer pruebas.

El otro aspecto por el cual suele ser complejo encarar este tipo de indicadores, es que una vez que tengamos "la data" luego hay que procesarla y no es lo mismo procesar una serie temporal de datos numéricos, por mas que haya que trabajar ciertas fallas, eso sería mucho mas simple que trabajar datos de sentimiento porque el lenguaje es bastante ambiguo por mas que no nos damos cuenta.

Esto pasa porque nuestro cerebro esta muy entrenado, luego de años, para interpretar lenguaje dentro de un contexto, esto es algo mucho mas difícil para un bot, les pongo un ejemplo, imaginen el siguiente tweet así de sencillo:

"\$SPY terrible imparable"

En una primera instancia en Argentina la palabra "terrible" tiene un sentimiento positivo, es raro, pero es así, imparable en ese contexto, reforzaría el sentimiento positivo como algo "super positivo". En otro país de habla hispana, sin embargo, la palabra "terrible" tiene un sentimiento negativo, de hecho si lo busco en la RAE encuentro estas dos acepciones:

1. adj. Que causa terror.
2. adj. Difícil de tolerar.

¿lo ven?

Supongamos que tenemos un barido de 100.000 tweets hablando en un determinado momento del SP500, no parece tan fácil ponerle puntajes o scores de sentimiento a las palabras así nomás. Y solo puse un ejemplo que se me ocurrió en el momento, este tema es realmente complejo.

Para quienes les interesa este tema de sentimiento, realmente es un tema que vale la pena, pero hay que dedicarle mucho tiempo y recursos, hay empresas dedicadas a estos temas, y es un campo que está en pleno desarrollo, generalmente se basan en sistema de IA enfocados en NLP (inteligencia artificial enfocada al procesamiento del lenguaje natural)

Pero cabe aclarar que el sentimiento de una expresión contiene por ejemplo "ironía", "humor" y un montón de cosas mas allá del "positivo o negativo" hay mil herramientas por ahí, la verdad, les soy sincero, no me gustó ninguna, pero como les digo esto es una industria super en pañales, estamos en sus inicios, por lo que no me extrañaría que en 2022 o 2023 ya haya buenos proveedores de APIs de análisis de sentimiento, sin embargo no los voy a dejar con las manos vacías, les dejo acá una mini guía de como usar la herramienta de GOOGLE, como sabrán de leerme soy fanático de sus servicios, en el t[5] les mostré como usar la API del traductor, y el uso de spreadsheets entre otras cosas, así que no me voy a detener en como abrir cuenta de servicios porque está en el t[5], solo les dejo esta secuencia de pasos:

Si algo cambió busquen en: <https://cloud.google.com/docs/authentication/getting-started>

Si sigue como hoy los pasos son:

1. En Cloud Console, ve a la página Crear cuenta de servicio.
2. Selecciona un proyecto
3. Ingresa un nombre en el campo Nombre de cuenta de servicio.
4. (el campo ID de cuenta de servicio lo completa solo google)
5. Clic en Crear y continuar.
6. Clic en el campo Seleccionar una función.
7. Clic en Acceso rápido => clic en Básica y, luego => en Propietario.
8. Clic en Continuar.
9. Clic en Listo para terminar de crear la cuenta de servicio.

Para crear una clave de cuenta de servicio:

1. En Cloud Console, clic en la dirección de correo electrónico de la cuenta de servicio que creaste.
2. Haga clic en Claves.
3. Haz clic en Agregar clave, luego haz clic en Crear clave nueva.
4. Haga clic en Crear. Se descargará un archivo de claves JSON en tu computadora.

Luego hay que ir a la API de NLP y habilitarla (por default está deshabilitada porque es de pago, pero Google a toda cuenta nueva le da USD 300 de prueba gratuitos, y pasado ese límite hay que autorizar explícitamente cualquier gasto, es confiable, lo expliqué en el t[5])

Para habilitar la API van a:

<https://console.cloud.google.com/apis/api/language.googleapis.com/overview>

O Naveguen desde el panel de Google Console hasta la API de NLP

El Json creado tiene nombres generados por Google a mi me creó algo así como:

spherical-xxxxxxxxx-294xx-8fxxxx238ee.json

Son palabras y números y caracteres sin mucho sentido, es un json con las claves privadas para firmar y autenticarse, lo tienen que poner en el mismo directorio donde lo van a usar.

Una vez hecho todo esto están en condiciones de poder usar la API de la IA de Google para medir el sentimiento de un párrafo, les dejo mi mini-implementación en Python

Antes que nada hay que instalar la librería de Google de NLP y proceder a las importaciones necesarias, en mi caso solo voy a usar esa librería y pandas para presentar los resultados:

```
# pip install google-cloud-language
import pandas as pd
from google.cloud import language_v1
```

Antes que nada debo "loguearme", hay muchas formas de hacerlo, lo mejor en mi opinión es declarar la constante como variable de entorno

```
import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "spherical-xxxxxxxxx-294xxx-8fxxxxx238ee.json"
```

Ahora armo una función que realice el trabajo y ya:

```
def analizar_texto(texto, language = "en"):
    client = language_v1.LanguageServiceClient()
    # Tipos posibles: PLAIN_TEXT, HTML
    type_ = language_v1.Document.Type.PLAIN_TEXT
    # Listado idiomas: https://cloud.google.com/natural-language/docs/languages
    document = {"content": texto, "type_": type_, "language": language}
    # Esto por los caracteres especiales: NONE, UTF8, UTF16, UTF32
    encoding_type = language_v1.EncodingType.UTF8
    req = {'document': document, 'encoding_type': encoding_type}
    response = client.analyze_sentiment(request = req)
    # Sentimiento de cada oración en el párrafo:
    results = []
    for sentence in response.sentences:
        r = {'oracion': sentence.text.content, 'puntaje':sentence.sentiment.score,
             'magnitud':sentence.sentiment.magnitude}
        results.append(r)
    return pd.DataFrame(results)
```

Como verán no todo es tan sencillo, tengo que resolver los siguientes temas:

- Predeterminar el tipo de enriquecido del párrafo (puede ser texto plano o algún lenguaje etiquetado como HTML)
- Detectar o predeterminar el idioma
- Detectar o predeterminar el “encoding” generalmente usamos UTF8 pero ojo que puede traer mucha complicación esto con los acentos y demás.
- Separar la estructura de oraciones por si hay divergencias en el sentimiento de cada oración

Bueno, de modo muy básico les resuelvo todo eso en la función citada como verán

Veamos entonces como anda esto con algunas frases:

```
frase = "Estoy super contento, que alegría. SPY terrible imparable"  
analizar_texto(frase, language="es")
```

	oracion	puntaje	magnitud
0	Estoy super contento, que alegría.	0.9	0.9
1	SPY terrible imparable	-0.3	0.3

Como ven por un lado analiza cada oración (separada por puntos) por separado y en forma independiente del resto de las oraciones que componen el párrafo

Por otro lado, vean que tiene una escala de -1 (totalmente negativo) a +1 (totalmente positivo) el puntaje y la magnitud es simplemente el valor absoluto del sentimiento.

Vean lo que les decía, lo de "SPY terrible imparable" lo detecta como algo negativo, sin embargo en Argentina esa frase tiene una denotación positiva, pero claro, el español es español, no se adapta esto a los modismos locales. Acá ya vemos con un ejemplo muy simple una falla importante.

Otro tipo de falla, es que esto no detecta ni el humor ni la ironía ni nada de eso, solo lee texto con una interpretación de contexto muy limitada, esperemos que esto avance con el tiempo, si no es así, ¡trabajemos para mejorarlo!

Miren otro ejemplo, vean como la palabra "desplome" no le asigna significado y vean como el resto de esa primera oración lo ve como positivo, sin embargo el "no aparecen compradores" ahí si le asigna un sentimiento negativo. Una táctica podría ser ir sumendo los puntajes siempre y cuando las magnitudes sean mayores o iguales a 0.5, cosa que no le de bola a las oraciones mas bien "neutra"

```
pd.options.display.max_colwidth = 80  
  
frase = """Desplome de los titulos Argentinos en las bolsas de europa continental.  
Titulos Argentinos en las bolsas de europa continental.  
No aparecen compradores"""  
  
analizar_texto(frase, language="es")
```

	oracion	puntaje	magnitud
0	Desplome de los titulos Argentinos en las bolsas de europa continental.	0.2	0.2
1	Titulos Argentinos en las bolsas de europa continental.	0.2	0.2
2	No aparecen compradores	-0.5	0.5

Como ven, no es tan sencillo como parece analizar en tiempo real de forma inequívoca, el sentimiento de miles de mensajes en redes sociales. De hecho, recién les mostré ejemplos (un poco a propósito reconozco) en donde falla esta evaluación automatizada.

¿A qué voy con esto?

Si bien el análisis de sentimiento (incluso provisto por APIs de empresas de primer nivel mundial que tienen una IA entrenada para esto) deja mucho que desear en el fin de una simple oración, la idea detrás de esto es analizar miles y miles de mensajes con el fin de "en el montón" detectar el verdadero sentimiento, más allá de la torpeza de la IA de NLP de Google para analizar una frase concreta.

Aclarado esto, como les dije, es una gran herramienta, y sobre todo es una gran industria en desarrollo que está generando grandes oportunidades, he visto análisis en varios ejes, no solo positivo y negativo, sino "nivel de ironía", "nivel de humor", "tristeza", "bronca", "amor" y cosas por el estilo, realmente un campo apasionante para investigar.

Lo más usual es investigar esto en twitter, por su inmediatez, pero también Reddit, Telegram, son otras redes donde se buscan datos de sentimiento.

Ratios y series de Análisis fundamental

Bueno, este tipo de datos, hay que tener presente que vienen con un retraso importante, es decir cuando sale un dato nuevo de balances, ya pasó más de un mes del cierre del trimestre, es decir es un dato que atrasa alrededor de 4 o 5 meses, sin embargo no solamente se considera dato de fundamental a los datos de balances, también hay determinados ratios o indicadores macro o sectoriales que se consideran "fundamentals"

Para resumir en un par de puntos podríamos agrupar este tipo de datos en:

- Indicadores macro de economía (empleo, actividad, sectoriales, IPC, etc)
- Indicadores sectoriales (por ejemplo, ratios de construcción a empresas del rubro, ratios de consumo masivo o comerciales a empresas de retail, datos turísticos a aerolíneas, etc)
- Estados contables
 - Cash Flow
 - Hoja de Balances
 - Estado de resultados
- Ratios intra estados contables
- Ratios entre estados contables e indicadores exógenos

Como les decía generalmente son muy tardios, así como los basados en series de precios tienen un leve retraso, y los de sentimiento son casi en tiempo real, en el caso de los "fundamentals" son más lentos, en el mejor de los casos son series mensuales que se conocen a los 15 días de terminado el mes, o sea 1.5 meses de delay, y hay casos de publicación trimestral con demoras de 2 meses o más para publicarse.

Básicamente esta dificultad de tener solo períodos muy altos respecto a los timeframes de pocos minutos en precios de los activos, nos lleva a dos soluciones:

- Usar los fundamentales solo para clasificar
- Interpolar los datos una serie mensual para llevarla por ejemplo a diaria

Lo más recomendable, al menos los mejores resultados que obtuve y vi en otros colegas es el primer caso, es decir, usar los fundamentales para clasificar regímenes, no como un indicador continuo, sino para, dentro de un marco de referencia decir si los fundamentales son buenos, neutros o malos, por decir algo y en función de ello utilizar diferentes método o parametrizaciones de los mismos.

Voy a hacer un ejemplo de este tipo de clasificación por fundamentales, solo a modo de ejemplo para que se vaya entendiendo la idea. Imaginen que vamos a utilizar un modelo de trading aplicado a alguno de los 500 activos del SP500, el método es bueno, pero creemos que funciona mejor con los activos que al momento presentan "Alto ratio P/S" es decir alto precio de mercado respecto al valor libros (es decir activos mejor valorados por el mercado que por sus balances)

¿Qué haríamos para backtestear esa hipótesis?

Bueno, backtestear y a eso llegaremos pronto, pero aquí en la sección "construcción de indicadores" construyamos la serie de datos que me clasifique cuales están con "Alto" o "Bajo" P/B

Bajemos primero los 500 tickers del SP500, forma práctica sin APIs ni nada (tomo los primeros 500, porque si son 503 creo pero la API de TDA no me trae más de 500 juntos)

```
sp500_wiki = pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')[0]
sp500_tickers = list(sp500_wiki.Symbol)

tickers = ','.join(sp500_tickers[:500])
tickers
```

```
ABE,ABT,ABC,ABVC,ACN,ATVI,ADBE,ADP,AEP,AES,AFL,A,APD,AKAM,ALK,ALB,ARE,ALGN,ALLE,LNT,ALL,GOOGL,GOOG,MO,AMZN,AMCR,AEE,AAL,AEP,A
APL,AZE,AMT,AIR,AIP,AIP,AIG,AMN,APN,APT,ANSS,APTN,AVX,ADS,APA,AAPL,APMT,APTV,ADM,ANET,AIG,ALZ,T,ATO,ADSK,ADP,AZO,AVB,AVY,BKR,BL
L,BAC,BBL,BAA,BCA,EROC,ESY,EISI,TECH,SLL,SLC,BC,BA,SONG,BAA,EXP,BAA,BMY,AVGO,BR,BF,B,CHRN,COG,CDNS,CZR,CPB,COF,CAH,KMX,CCL,CA
PA,CTU,CAT,CBRE,CBRE,CDC,CE,CNC,CMP,CFNL,CF,CRL,SCM,CHTR,CVN,CNG,CB,CHO,CI,CINF,CTAS,CSCO,C,CFG,CTXS,CLX,CME,CMS,KO,CTSH,CL,C
WOR,CNA,CAG,COP,ED,STZ,COO,CPT,EW,CTVA,COST,CCL,CXA,CRL,CVS,DHL,DHR,DRI,DVA,DE,DAL,XRN,DVN,DXM,FANG,DLR,DFS,DISCA,DISCK,DI
SI,DLB,DTRE,C,DPZ,DM,DUW,OTE,DUK,CRE,DO,DNC,ENN,ETN,EBAY,ECL,EIX,EW,EA,EMR,ENPH,ETR,EOG,EFX,EQIX,QOR,ESS,EL,ETSY,EVRG,ES,RE,EX
C,LNP,EXP,EXS,NOM,FFIV,FS,FAST,FRT,FDX,FIG,FIG,FE,FRV,FLSV,FLT,FMCI,F,FTNT,FTV,FBHS,FOXA,FOX,BEN,FCX,GPS,GRMI,IT,GNRC,GO,GE,G
TS,ISK,SPC,SELU,SL,SPM,SS,BAN,HAL,HBI,HIZ,HAS,HCA,PEAK,HSIC,HSY,HS,HP,E,HOLX,HO,HON,HRL,HST,HMM,HPQ,HUM,HBII,HII,IEX,IDX,IN
FC,ITAT,JOHN,JOH,JIR,INTU,ISRG,IVZ,LPGP,LQV,URN,JND,J,T,JBHT,STM,IND,JCI,JPML,INPR,KSU,K,KEY,KEYS,KNB,KIN,
CIE,CIAZ,KHC,KS,LA,LCR,LLN,LVS,LEG,LOOS,LEN,LVLY,LNC,LLN,LVY,LXQ,LMT,L,LOW,LUMN,LVB,MTB,MRO,MPG,MKTX,MAR,MMC,MIM,MAS,MA,MKC,
MCFC,MCX,MOT,MPC,MTD,MSP,MCP,MFC,MISI,MIA,MKA,MHK,TAP,MDLZ,MPWR,MIST,PCO,MS,NS,NS,NSI,MSCL,NDAQ,NTAP,NFLX,NWL,NEM,NWSA,MWS,NE
E,MSL,VN,NEK,RL,NSC,NTR,S,NOV,NOK,NCLH,NOV,NRG,ORL,NVDA,NVR,NXP,ORL,ORLY,OXY,COFL,OVC,OKC,ORL,OGN,OTIS,PCAR,PKG,PH,PAX,PAWC,PYPL,P
EW,PUR,PSCT,PEP,PKC,PRO,PFE,PM,PSX,PNW,PXG,PPL,PFG,PG,PGR,PLD,PRU,PTC,PEG,PSA,PHM,PVH,QRVO,PWR,QQCOM,DGX,RL,RJF,R
TSL,RSB,REGN,RF,RSG,RMD,RHT,ROK,ROL,ROP,ROST,RCL,SPGI,CRP,SBAC,SLB,STX,SEE,SRE,NOW,SHV,SPG,SXKS,SNA,SO,LUV,SKK,SBUX,STI,STE,SY
E,STVE,SYS,SNPS,SYY,TBLS,TROW,TTWO,TPR,TGT,TEL,TDY,TFX,TER,TSIA,TXN,TMO,TDX,TSCO,TT,TDG,TRV,TRMB,TFC,TWTR,TYL,TSN,UDR,ULTA,
USE,VAL,VA,UNP,VAL,UHS,UPS,URI,LHS,UNP,VLO,VTR,VRSN,VRSK,VZ,VRTX,VFC,VIAC,VTRS,V,VNO,VMC,WRB,WAB,WMT,WBA,DIS,WM,WAT,WE
C,WFC,WL,WMC,WY,WRK,WRS,WLT,WYNN,XEL,XLUK,XYL
```

Los dejo en un solo string separados por coma porque así los necesito en la API de TDA Ameritrade para bajarlos de fundamentales en un batch request (ver t[3] y t[4])

Una vez que tengo esto, armemos la función para bajar los fundamentales de la API de TDA Ameritrade

```
import requests
c_key = 'AHSA7C8FWZZFRG1W15N0JLQY8WRSCOVI'

def fundamentals(symbols, pr='fundamental'):
    params = {'apikey' : c_key, 'symbol':symbols, 'projection': pr}
    endpoint = 'https://api.tdameritrade.com/v1/instruments'
    r = requests.get(url=endpoint ,params=params)
    return r.json()
```

Obviamente saquen su api_key en: <https://developer.tdameritrade.com/apis>

Y ahora que tengo la función preparada le mandamos el request de una para los 500 activos:

```
data = fundamentals(tickers)

lista = []
for ticker in tickers.split(","):
    lista.append(data[ticker]['fundamental'])

df = pd.DataFrame(lista).set_index('symbol')
```

symbol	high52	low52	dividendAmount	dividendYield	dividendDate	dividendPayDate	beta	vol1DayAvg	vol10DayAvg	vol3MonthAvg
MMM	208.9500	156.1300	5.92	3.26	2021-08-20 00:00:00.000	2021-09-12 00:00:00.000	0.96267	4026420.0	4047577.0	50777550.0
ABT	129.6992	100.3400	1.80	1.42	2021-10-14 00:00:00.000	2021-11-15 00:00:00.000	0.64801	4515490.0	4519978.0	99290890.0
ABBV	121.5300	79.1101	5.20	4.83	2021-10-14 00:00:00.000	2021-11-15 00:00:00.000	0.81092	8710040.0	8777595.0	146998140.0
ABMD	387.4000	242.7300	0.00	0.00	1.32658	191780.0	191801.0	4191430.0
ACN	345.5199	210.4200	3.52	1.05	2021-07-14 00:00:00.000	2021-08-13 00:00:00.000	1.11750	1966970.0	1971181.0	37596670.0
...
WLTW	271.8700	179.3100	3.20	1.39	2021-09-29 00:00:00.000	2021-10-15 00:00:00.000	0.80251	2284940.0	2285057.0	38715900.0
WYNN	143.8800	67.7000	0.00	0.00	...	2020-03-05 00:00:00.000	2.45612	9180840.0	9183861.0	87460950.0
XEL	76.4400	57.2300	1.83	2.86	2021-09-14 00:00:00.000	2021-10-20 00:00:00.000	0.31734	2929830.0	2929869.0	49840000.0
XLNX	159.3000	96.7100	0.00	0.00	...	2020-12-02 00:00:00.000	1.00789	1585880.0	1588150.0	47918210.0
XYL	138.7800	80.9500	1.12	0.85	2021-08-25 00:00:00.000	2021-09-23 00:00:00.000	1.04259	764250.0	767020.0	16469750.0

500 rows x 45 columns

Y ahora lo importante, necesitamos un indicador que nos diga en cada activo si tiene ALTO o BAJO valor de P/B, separamos por la mediana por ejemplo y tenemos:

```
df.loc[df.pbRatio < df.pbRatio.median(), 'pb_ratio_tipo'] = 'Bajo'
df.loc[df.pbRatio > df.pbRatio.median(), 'pb_ratio_tipo'] = 'Alto'
df
```

symbol	high52	low52	dividendAmount	dividendYield	dividendDate	beta	vol1DayAvg	vol10DaysAvg	volMonthAvg	pb_ratio_tipo
MMM	208.9500	156.1300	5.92	0.26	2021-09-20 00:00:00.000	0.96267	4026420.0	4047677.0	50777560.0	Abs
ABT	129.6302	100.3400	1.80	1.42	2021-10-14 00:00:00.000	0.64801	4615490.0	4619678.0	59290890.0	Abs
ABBV	121.5300	79.1101	5.20	4.83	2021-10-14 00:00:00.000	0.81092	6710040.0	6777595.0	146996140.0	Abs
ASMD	387.4000	242.7300	0.00	0.00		1.32658	191780.0	191801.0	4191430.0	Abs
ACN	345.5100	210.4200	3.52	1.05	2021-07-14 00:00:00.000	1.11750	1966970.0	1971181.0	37596670.0	Abs
—	—	—	—	—	—	—	—	—	—	—
WLTW	271.8700	179.3100	3.20	1.39	2021-09-29 00:00:00.000	0.80261	2284940.0	2285057.0	38715900.0	Bajo
WYNN	143.5800	67.7000	0.00	0.00		2.46612	9180840.0	9183861.0	87490950.0	Abs
KEL	76.4400	57.2300	1.83	2.98	2021-09-14 00:00:00.000	0.31734	2929830.0	2929869.0	49840000.0	Bajo
XLNX	159.3000	96.7100	0.00	0.00		1.00789	1585880.0	1588160.0	47916210.0	Abs
XYL	138.7800	80.9500	1.12	0.65	2021-08-25 00:00:00.000	1.04259	764260.0	767020.0	16469750.0	Abs

500 rows x 11 columns

Fijense que interesante que la mitad de P/B Bajo, tiene mas alto Beta, es decir están ahora (sept 2021) notablemente más sensibles al movimiento del mercado que los de P/B Alto.

```
df.groupby('pb_ratio_tipo').beta.mean()

pb_ratio_tipo
Alto    1.041800
Bajo   1.161464
Name: beta, dtype: float64
```

Exógenos, externos

Por último, quedan los datos totalmente exógenos a la empresa o activo a operar, acá hay datos de todo tipo, puede ser algo económico o medianamente relacionado, pero también pueden ser datos que no tienen nada que ver como datos del clima, del tránsito, incluso datos de cámaras de seguridad que realizan conteos de gente entrando o saliendo de shopings etc, obviamente son cosas muy específicas, solo las menciono para que las tengan en cuenta:

- Clima
- Tránsito, Autopistas
- Info deportiva, eventos, tickets, consumos
- Info de vuelos
- Mediciones concretas de un comercio (tickets, kpis de campañas, etc)
- Info de cámaras de seguridad

Claramente la desventaja de este tipo de datos como indicadores es que son muchísimo mas difíciles de conseguir, mientras que por el otro lado la gran ventaja es que, al ser datos que probablemente nadie use en sus modelos, dan un valor agregado que puede marcar la diferencia.

Ejercicios de Construcción de indicadores

- 1- Construir el indicador Oscilador de Chaikin basado en las siguientes fórmulas:

$$N = \frac{(Close - Low) - (High - Close)}{High - Low}$$

M = N * Volume (Period)

ADL = M (Period - 1) + M (Period)

CO = (3-day EMA of ADL) - (10-day EMA of ADL)

Siendo:

N = Money flow multiplier

M = Money flow volume

ADL = Accumulation distribution line

CO = Chaikin oscillator

Construirlo con TSLA bajando datos hasta el 4 de septiembre de 2021

- 2- Una vez construido ese oscilador, acotarlo o transformarlo en oscilador acotado usando el método del zscore dentro de una ventana de 100 velas.

Graficar la construcción acotada por zscore de dicho oscilador

- 3- Y Calcular el R2 de correlación con forwards a diferente cantidad de velas entre 1 y 100, y graficar esa evolución de los R² para encontrar la mejor ventana de correlación de variación futura vs el indicador construido.

- 4- Construir un simple oscilador del precio de cierre de TSLA en una ventana de 100 velas

- En función del rango min/max
- En función del zscore
- Construir ambos en una escala acotada entre 0-1
- Graficar

- 5- Construir para TSLA un indicador discreto que contabilice la cantidad de velas que abren con GAP > 1% en una ventana de 100 velas y buscar el valor de ventana fw óptimo para el cual la correlación medida por el r² entre el indicador y el forward es mayor

Respuestas a ejercicios de indicadores

```
#=====
#      Ejercicio 1      #
=====

import yfinance as yf
import pandas as pd

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04', progress=False)
fast, slow = 3,10

data['MFMult'] = (2*data.Close-data.Low-data.High) / (data.High-data.Low)
data['MFVol'] = data['MFMult'] * data.Volume
data['ADL'] = data['MFVol'].cumsum()
data['CO'] = data['ADL'].ewm(span=fast).mean() - data['ADL'].ewm(span=slow).mean()

data.dropna()
```

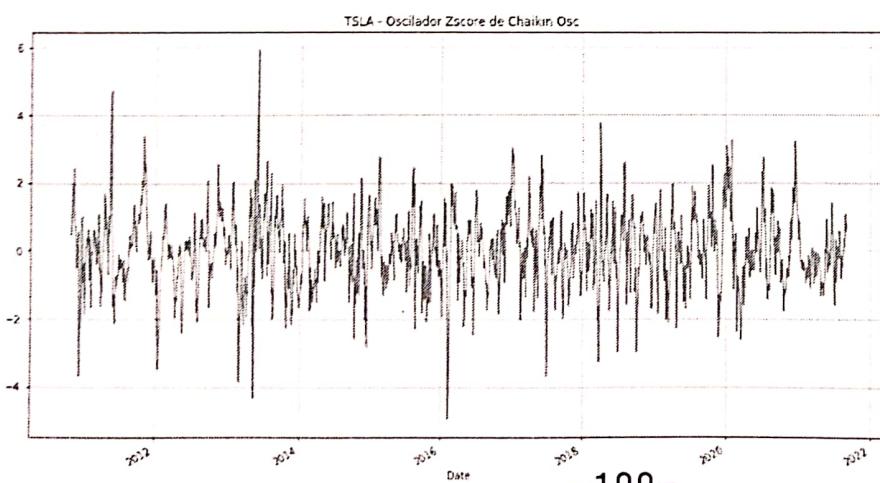
Date	Open	High	Low	Close	Volume	MFMult	MFVol	ADL	CO
2010-06-29	3.800000	5.000000	3.508000	4.778000	93831500	0.702413	6.590844e+07	6.590844e+07	0.000000e+00
2010-06-30	5.152000	6.084000	4.660000	4.766000	85935500	-0.851124	-7.314174e+07	-7.233301e+06	-8.533203e+06
2010-07-01	5.000000	5.184000	4.054000	4.392000	41094000	-0.401769	-1.651031e+07	-2.374361e+07	-1.203127e+07
2010-07-02	4.600000	4.620000	3.742000	3.840000	25699000	-0.776766	-1.996210e+07	-4.370572e+07	-1.571023e+07
2010-07-06	4.000000	4.000000	3.166000	3.222000	34334500	-0.865707	-2.972363e+07	-7.342935e+07	-2.201570e+07
...
2021-08-30	714.719971	731.000000	712.729980	730.909973	18604200	0.990145	1.842085e+07	3.500913e+09	8.388540e+06
2021-08-31	733.000000	740.390015	726.440002	735.719971	20855400	0.330460	6.891880e+06	3.507805e+09	1.231383e+07
2021-09-01	734.080017	741.989990	731.270020	734.090027	13204300	-0.473878	-6.257225e+06	3.501548e+09	1.080925e+07
2021-09-02	734.500000	740.969971	730.539978	732.390015	12777300	-0.645247	-8.244512e+06	3.493304e+09	6.587834e+06
2021-09-03	732.250000	734.000000	724.200012	733.570007	15246100	0.912246	1.390820e+07	3.507212e+09	8.687331e+06

2817 rows x 9 columns

```
#=====
#      Ejercicio 2      #
=====

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04', progress=False)
fast, slow = 3,10

data['MFMult'] = (2*data.Close-data.Low-data.High) / (data.High-data.Low)
data['MFVol'] = data['MFMult'] * data.Volume
data['ADL'] = data['MFVol'].cumsum()
data['CO'] = data['ADL'].ewm(span=fast).mean() - data['ADL'].ewm(span=slow).mean()
data['CO_pct'] = (data['CO'] - data['CO'].rolling(100).mean()) / data['CO'].rolling(100).std()
data['CO_pct'].plot(figsize=(15,8), grid=True)
```



```

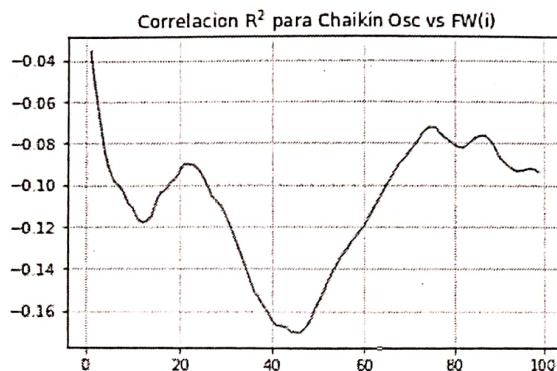
#=====
#      Ejercicio 3      #
#=====

corrs = []
for i in range(1,100):
    data['fw'] = data.Close.shift(-i) / data.Close
    corr_p = data['CO_pct'].corr(data.fw)
    corrs.append(corr_p)

s = pd.Series(corrs, index=range(1,100))
s.plot(title=r'Correlacion R$^2$ para Chaikin Osc vs FW(i)', grid=True)
print(f"FW óptimo para Correlación: {s.idxmin()}\nR2={s.min():.4f}")

```

FW óptimo para Correlación: 45
R2=-0.1749



```

#=====
#      Ejercicio 4      #
#=====

import yfinance as yf
import pandas as pd

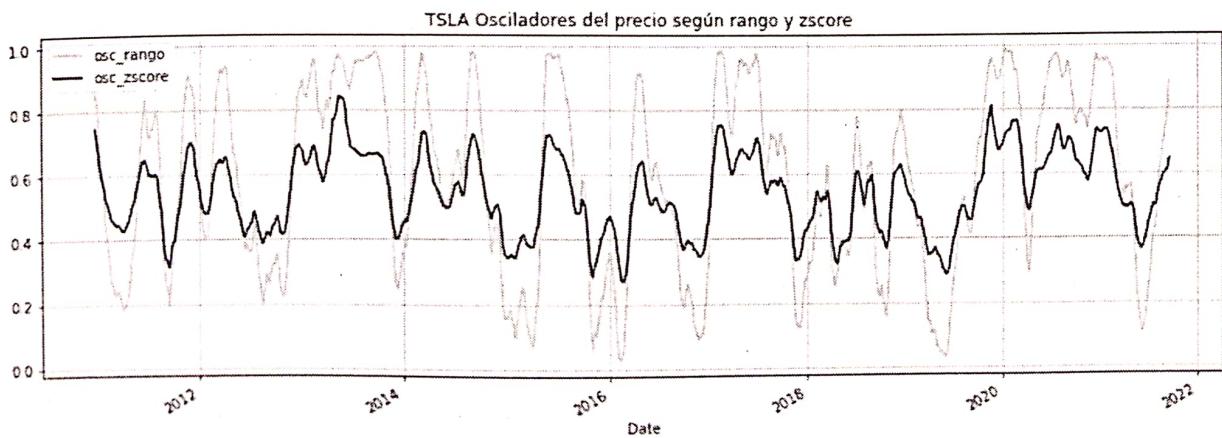
data = yf.download('TSLA', auto_adjust=True)

osc_min = data.Close.rolling(100).min()
osc_max = data.Close.rolling(100).max()
osc_mean = data.Close.rolling(100).mean()
osc_std = data.Close.rolling(100).std()

data['osc_rango'] = (data.Close - osc_min) / (osc_max - osc_min)
data['osc_zscore'] = (data.Close - osc_mean) / osc_std
data['osc_zscore'] = (data['osc_zscore']) / data['osc_zscore'].max() + 1/2

data.iloc[:, -2:].rolling(20).mean().plot(figsize=(15,5), grid=True)

```

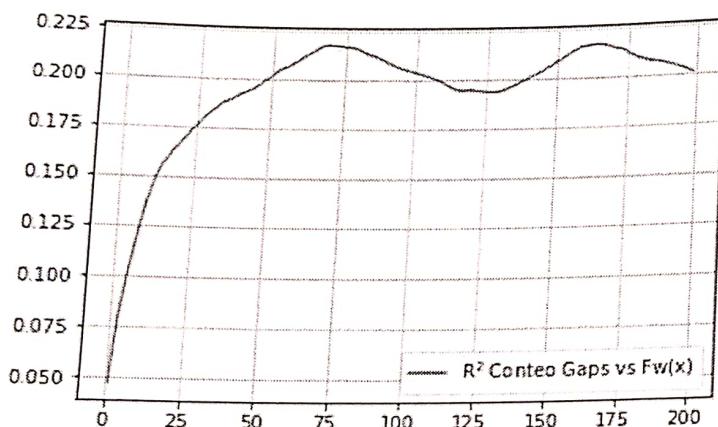


```

#-----#
#          Ejercicio 5          #
#-----#
import yfinance as yf
import numpy as np
import pandas as pd
data = yf.download('TSLA', auto_adjust=True, end='2021-09-04')
w = 100
data['tipo_gap'] = np.where(data.Open / data.Close.shift() > 1.01, 1, 0)
data['conteo_gap'] = data['tipo_gap'].rolling(w).sum() / w
rango_fw = range(200)
corrs = []
for i in rango_fw:
    data['fw'] = data.Close.shift(-i) / data.Close - 1
    c = data.conteo_gap.dropna().corr(data.fw)
    corrs.append(c)
df = pd.DataFrame(corrs, index=rango_fw, columns=[r'R$^2$ Conteo Gaps vs Fw(x)'])
df.plot(grid=True)
optimo = df.idxmax().values[0]
print(f"\nEl valor óptimo de ventana FW para maximizar correlación es {optimo} velas FW\n")

```

El valor óptimo de ventana FW para maximizar correlación es 69 velas FW



Filtro de datos

Y llega la parte mas aburrida de la etapa de preBacktest que tiene que ver con el filtro de los datos, acá hay todo tipo de cuestión a verificar, les enumero las fallas más típicas en series de precios

1. No tienen incluidos los dividendos
2. Tienen ajuste por dividendos pero solo en los precios de cierre y no en el OHLC
3. Tienen ajuste por dividendos pero no por splits
4. Tienen incluidos los ajustes por dividendos o splits pero hay un Split no registrado
5. Tienen todo un período sin datos (cambios de ticker etc)
6. Tienen un período donde pasa de un precio y régimen de volatilidad a otro que nada que ver
7. Tienen spikes muy fuera de rango
8. La base tomada de referencia tiene sesgos que no eran deseados inicialmente
9. Otros sesgos: Son todos activos de un mismo sector
10. Otros sesgos: Son todos High/low Beta
11. Otros sesgos: Survivorship Bias

Tema de dividendos

Claramente si voy a backtestear algo, la idea es que mientras esté comprado el efecto dividendo no afecte al backtest, es decir que un activo que paga dividendos no sea afectado por este hecho, para ello se trabaja con las series limpias, es decir con los precios ajustados, es decir, que si un activo cotiza al cierre a \$100 y paga dividendos de \$4, al otro día abre a \$96, pero ese \$96 post pago de dividendos es igual al \$100 antes de pagarlos, es decir, si un trader estaba comprado, no perdió los \$4 porque les entraron como dividendos. Para solucionar esto, lo que se hace es partir del último precio spot e ir descontando (ajustando) hacia atrás de modo tal que los dividendos se compensen y quede una serie uniforme

En el caso de yfinance ya tiene resuelto el tema dividendos con el parámetro `auto_adjust=True` y el tema `splits` por default los limpia, aunque ojo que a veces si consultan el dia del Split puede que no lo tenga ajustado, es un problema esto porque les puede hacer saltar alarmas automatizadas falsas porque no limpia un Split. Lo mejor como siempre es asegurarse de hacer la limpieza uno mismo

Aclaración fiscal: Como verán mas adelante, haré caso omiso a temas como la incidencia fiscal de los dividendos respecto a un mismo payoff dentro del precio. Claramente no es lo mismo, es decir, tener en cartera un activo que siempre vale \$100, no es lo mismo que tener en cartera uno que pasa de valer 100 a 96 y que nos deja un dividendo de \$4 en la cuenta en efectivo. Mas allá del tema fiscal hay varias cosas a tener en cuenta respecto a la reinversión o no del mismo, pero como les anticipé al principio la ideal del backtesting no es predecir el resultado futuro, sino validar una idea, comprar resultados, parametrizar métodos etc, con lo cual estos efectos solemos despreciarlos para simplificar el análisis.

Tema de Splits

Ojo, generalmente las buenas fuente de datos tienen incluido el descuento de todos los splits en los precios ajustados, eso no es problema, a menos que la API o fuente de datos sea muy precaria, pero ojo que hay muchas pero muchas veces que ajustan los cierres o a veces los OHLC por splits pero a veces no hacen lo mismo con el volumen nominal, y esto puede generar saltos muy absurdos en los volúmenes nacionales o cualquier indicador que deriven del mismo, me explico..

Supongan que tienen un activo que cotiza a \$100, hace un Split de 5 a 1, y pasa a cotizar \$20, si antes se negociaban por ejemplo 500 nominales por día cuando valía \$100, ahora que vale \$20, esos 500 nominales equivalen a 2500 nominales pre/split pero si la serie de precios está ajustada, debo multiplicar por una serie uniforme de nominales también y no por una con los saltos.

Veamos el ejemplo con la API de alphaVantage

```
import requests
apikey = '2RG2NEF3IPXMIPX3'

def alpha_download(symbol, size="full", start=None):
    function='TIME_SERIES_DAILY_ADJUSTED'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function' : function, 'symbol': symbol,
                  'outputsize': size, 'apikey': apikey}

    r = requests.get(url, params=parametros)
    data = r.json()['Time Series (Daily)']
    df = pd.DataFrame.from_dict(data, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df.columns = ['Open','High','Low','Close','AdjClose','Volume','Div','Split']
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)

    if start:
        df = df.loc[df.index >= start].copy()
    return df
```

Ya tenemos la función para bajar precios ajustados de cierre, veamos cuando hubo splits:

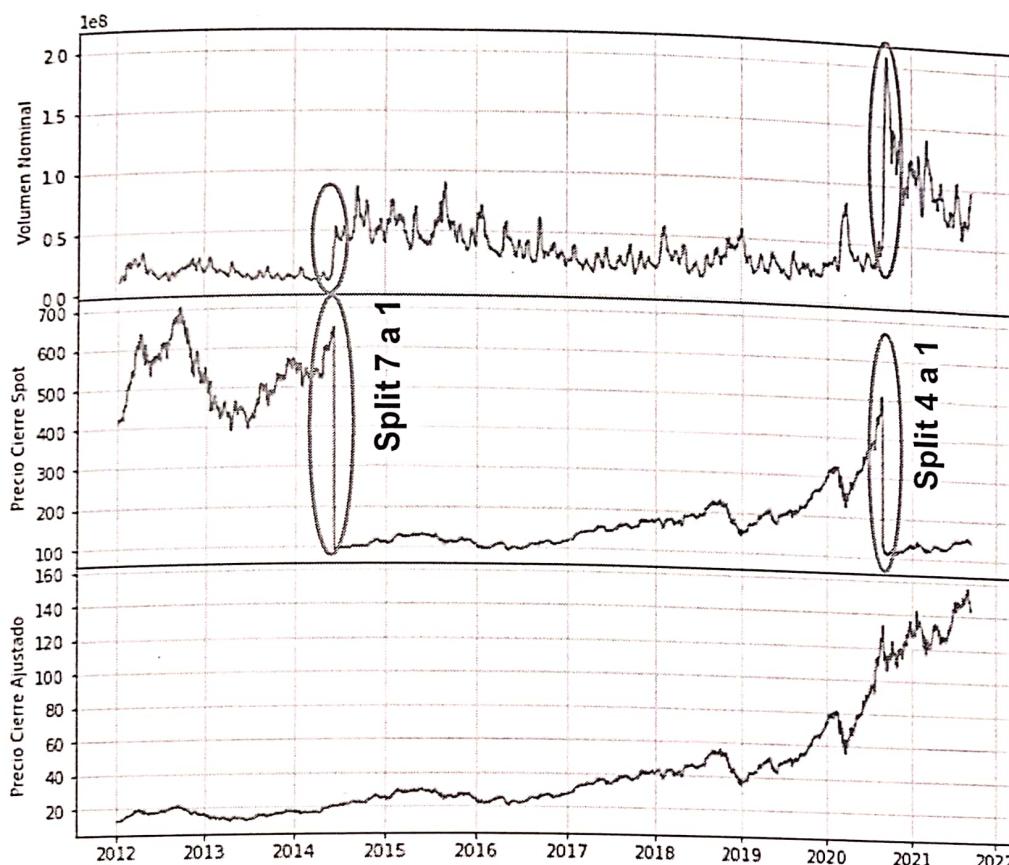
```
data_a = alpha_download('AAPL', start='2012-01-01')
data_a.loc[data_a.Split != 1]
```

Date	Open	High	Low	Close	AdjClose	Volume	Div	Split
2014-06-09	92.70	93.88	91.75	93.70	21.04	75414997.0	0.0	7.0
2020-08-31	127.58	131.00	126.00	129.04	128.21	223505733.0	0.0	4.0

Como ven, un Split de 7 a 1 en 2014, y otro de 4 a 1 en 2020, veamos lo que les comentaba, la serie ajustada de cierres y la serie de volumen:

Se los muestro en un gráfico porque se aprecia mucho mejor:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(10,9), nrows=3)
ax[0].plot(data_a.Volume.rolling(10).mean())
ax[0].set_ylabel('Volumen Nominal')
ax[1].plot(data_a.Close)
ax[1].set_ylabel('Precio Cierre Spot')
ax[2].plot(data_a.AdjClose)
ax[2].set_ylabel('Precio Cierre Ajustado')
[ax[i].grid() for i in range(3)]
plt.subplots_adjust(hspace=0)
```



¿Ven lo que les digo?

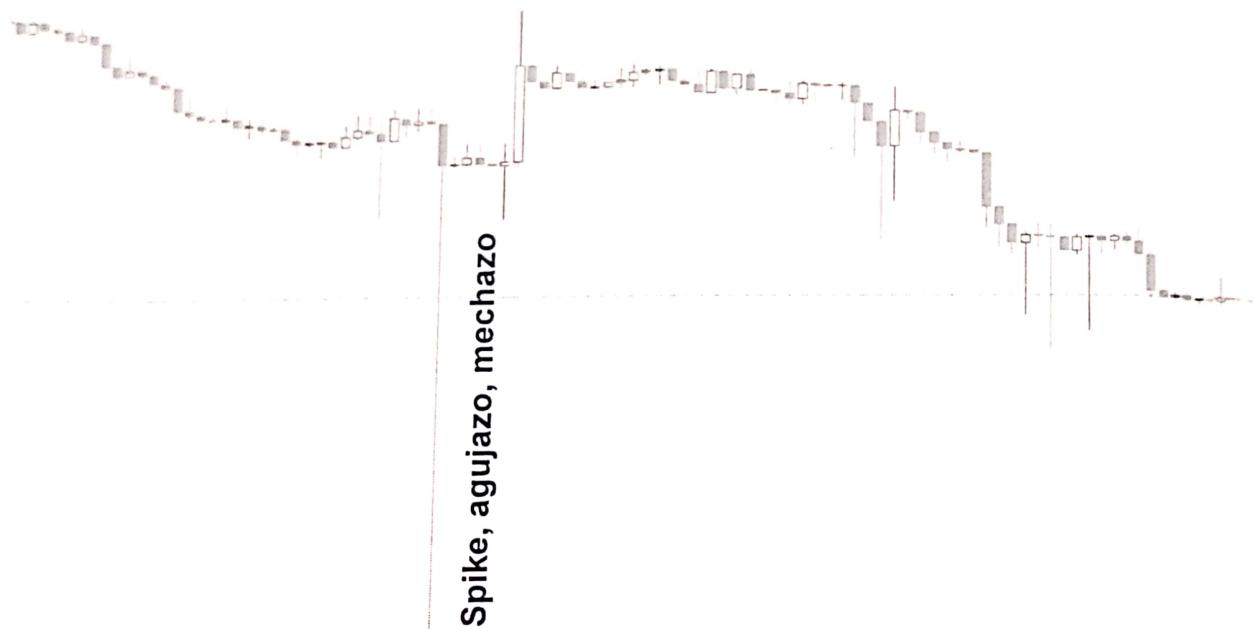
Si nos confiábamos y usábamos la serie ajustada de precio con la serie de volumen para calcular el nocional en millones de USD negociados por día, hubiéramos cometido un error fuerte, ya que el volumen nominal está sin ajustar, es decir, en cantidad de nominales del spot de cada día sin tener en cuenta el Split. Si usamos este volumen para calcular el nocional debemos usar entonces el precio sin ajustar, y si usamos la serie de precios ajustada debemos ajustar entonces el volumen.

No me voy a poner a mostrarles un ejemplo de cada cosa, pero creo que se entiende mi idea, siempre revisen bien los datos, una forma rápida es tirar un gráfico porque es visual y cómodo para entender si los datos que estamos usando están bien o las series presentan "saltos" que no esperábamos.

Spikes (aguajazos)

Otro caso típico de dato a limpiar son los "spikes", son lo que en Argentina decimos "dedazos" o dicho de otra forma "operaciones muy fuera de rango normal de negociación" que por lo general son de muy bajo volumen, y en momentos de muy baja liquidez, también pueden ser errores en la serie o alguna otra cosa así, estos datos deben descartarse (a menos que hagamos un bot especialmente diseñado para operar esas cosas, pero es raro porque no pasan casi nunca)

Ahora el tema es que si bien no pasan casi nunca, cada tanto ocurren y pueden deformar mucho una métrica que usa ese dato. Supongan que quedan abiertas las puntas con super bajas cantidades tanto en el bid como el ask, y de repente entra una orden a mercado de mucho mas cantidad que lo que hay en las puntas. ¿Qué pasa? Para que esa orden barre toda la cola de bids y asks y genera esos "mechazos" que en seguida se arbitran y aparecen las puntas o market makers y vuelve la serie a la normalidad pero habiendo generado una vela totalmente atípica, que, si usamos un precio promedio OHLC o alguna cosa así en un indicador, va a deformar toda la serie pudiendo generar grandes efectos cuando es algo insignificante que debimos haber descartado inicialmente



Ahora, imaginen que tienen que chequear si esta todo ok luego de bajarse las series de 1000 activos ¿ni locos nos?

Bueno, ahí tienen que buscar formas mas analíticas de buscar estas cosas, y una buena forma pueden ser los zscores o bien los % de mechas sobre la volatilidad, esto último es adaptativo al contexto, es decir es esperable que por mas mechazo normal que tenga, no debe ser mas de 5 a 10 veces la volatilidad normal de las ultimas por ejemplo 20 velas.

A modo de ejemplo va código de como calcular esto

Primero vamos a calcular los % de mechas tanto superior como inferior, luego la volatilidad de las ultimas 20 velas, y finalmente hacemos el cociente

```

import yfinance as yf
import numpy as np

data = yf.download('GGAL', auto_adjust=True)
data['mecha_sup'] = np.where(data.Close > data.Open, data.High/data.Close-1, data.High/data.Open-1)
data['mecha_inf'] = np.where(data.Close < data.Open, 1-data.Low/data.Close, 1-data.Low/data.Open)
data['volatilidad'] = data.Close.pct_change().rolling(20).std()
data['mecha_sup_aj'] = data['mecha_sup']/data['volatilidad']
data['mecha_inf_aj'] = data['mecha_inf']/data['volatilidad']

```

Una vez tenemos esto vemos las primeras 5 superiores:

```

data['mecha_sup_aj'].sort_values(ascending=False).head()

Date
2012-06-11    6.155715
2009-01-21    5.791319
2010-10-27    5.209250
2008-10-22    4.227832
2010-08-12    4.221667
Name: mecha_sup_aj, dtype: float64

```

Y las 5 mechas inferiores más grandes:

```

data['mecha_inf_aj'].sort_values(ascending=False).head()

Date
2006-12-21    5.207458
2007-04-13    4.720432
2001-07-12    4.340681
2014-10-15    4.045729
2005-02-02    3.867371
Name: mecha_inf_aj, dtype: float64

```

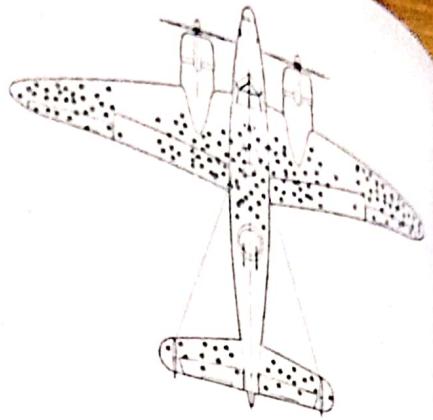
Como verán para GGAL en esas fechas, son mechas dentro de todo normales, mas de 10 veces la volatilidad ya deberíamos clipear() el dato (recortar)

Sesgos (supervivencia y selección)

Bueno, finalmente viene el tema de los sesgos, es muy importante esto, y ya no pasa por limpiar los datos tal cual, sino más bien por “pensar” los datos, por ejemplo, el error mas común es backtestear una estrategia de swing trading con las 500 del SP500, el problema es el siguiente, si nuestra estrategia es del tipo trend-following, al usar solo las empresas que sobrevivieron y hoy siguen en el índice, estamos usando empresas que fueron exitosas (o al menos, no usamos las que fueron a la quiebra o les fue muy mal y fueron deslistadas del índice)

Esa trampa es conocida como el “Survivorship Bias” o sesgo de supervivencia, no se si será verdad pero en todos los libros cuentan la historia que en la 2° guerra mundial hicieron un informe de ingeniería de daños causados a los aviones en combate, y viendo las zonas mas dañadas, deciden reforzarlas

Resumiendo, la cosa es que los aviones que usan para el estudio son los que vuelven llenos de balas, es decir los que sobreviven a los ataques, por lo tanto están reforzando las zonas mas resistentes, y no las que hacen que los aviones caigan, de las zonas débiles (cabina y turbinas) no se enteran porque no llegan esos aviones para ser estudiados ya que cuando dan las balas en esas zonas el avión por lo general cae en combate



Algo muy similar nos puede pasar cuando hacemos backtesting con las 500 empresas del SP500, esas 500 empresas son los aviones sobrevivientes, es decir las 500 empresas que siguieron estando en el SP500, pero allí no estarán ni BlackBerry, ni BlockBuster, ni Kodak o ninguna de aquellas empresas que fueron furor en su momento y luego cayeron en combate.

Bueno, el sesgo de supervivencia, es un caso especial del sesgo de selección, estos errores se ven muy claro en las encuestas electorales, o por ejemplo si estamos estimando la altura de la población y para ello hacemos una encuesta y nos paramos en la puerta de un club de Basketball claramente vamos a tener una muestra sesgada por gente mucho mas alta que lo habitual que generalmente suele practicar ese deporte.

Este tipo de sesgos a veces es medio difícil de darse cuenta, aquí podríamos aplicar una validación cruzada, en donde podríamos validar los resultados del backtest con otras muestras para chequear que sean similares, si encontramos grandes diferencias, es probable que la muestra elegida haya estado sesgada.

Introspección de Features

Bien, llegamos al punto en que ya tenemos varios features o indicadores para nuestro bot de trading algorítmico, ya comprobamos que los datos están limpios y no tienen sesgos, pero nos falta aún algunas cositas mas a evaluar las más relevantes:

- 1- ¿Qué tan interesante es el R^2 de cada indicador contra el target a backtestear?
- 2- ¿Hay multicolinealidad? ¿Se superponen diferentes indicadores? (inflación de varianza)
- 3- ¿Hay autoregresión en los features?
- 4- ¿Los features tienen un comportamiento normal? ¿y El target? ¿contra que dist fitean?
- 5- ¿Está balanceada la muestra?

Son algunas de las preguntas de data profiling que uno se suele hacer, a veces esto nos permite concentrarnos mejor en los indicadores que realmente nos aportan valor o que menos confusión pueden generar, ya veremos.

Para empezar, diremos que el primer punto a revisar es contrastar todos los indicadores que tengamos contra el mismo "target" a backtestear para ver cuál es más relevante.

Esto del target dependerá del tipo de bot, para simplificar uso el ejemplo del bot de trend-following que con algunos indicadores simples intentará darme alguna ventaja estadística en la predicción de si en "X" velas adelante es mas probable que suba o baje el activo en cuestión.

Así que vamos a armar una serie de features solo a modo de ejemplo:

```
import yfinance as yf, numpy as np

# Hiperparametro => para los rollings
n = 100

# Hiperparametro => para el RSI
n_rsi = 20

# Hiperparametro => para los cruces
fast1, slow1 = 20, 60
fast2, slow2 = 25, 65

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04', progress=False)
data['Balance'] = np.where(data.Close>data.Close.shift(), data['Volume'],
                           np.where(data.Close < data.Close.shift(), -data['Volume'], 0))

data['OBV'] = data['Balance'].cumsum()
dif = data['Close'].diff()
win = pd.DataFrame(np.where(dif > 0, dif, 0), index=data.index)
loss = pd.DataFrame(np.where(dif < 0, abs(dif), 0), index=data.index)
ema_win = win.ewm(alpha=1/n_rsi).mean()
ema_loss = loss.ewm(alpha=1/n_rsi).mean()
rs = ema_win / ema_loss

# Features
data['cruce1'] = data.Close.rolling(fast1).mean() / data.Close.rolling(slow1).mean() -1
data['cruce2'] = data.Close.rolling(fast2).mean() / data.Close.rolling(slow2).mean() -1
data['rsi'] = 100 - (100 / (1+rs))
data['sigma'] = data.Close.pct_change().rolling(n).std()
data['skew'] = data.Close.pct_change().rolling(n).skew()
data['kurtosis'] = data.Close.pct_change().rolling(n).kurt()
data['OBV_osc'] = (data.OBV - data.OBV.rolling(n).mean()) / (data.OBV.rolling(n).std())
features = data.iloc[:, -7:].dropna()
features
```

Date	cruce1	cruce2	rsi	sigma	skew	kurtosis	OBV_osc
2010-11-18	0.130694	0.100337	70.085913	0.045694	0.388374	4.494572	1.810285
2010-11-19	0.144293	0.109956	72.163427	0.045813	0.363155	4.405848	2.312055
2010-11-22	0.160633	0.122682	76.006381	0.045648	0.366442	4.417537	2.719116
2010-11-23	0.177235	0.137458	77.587571	0.043783	0.581596	4.828015	3.021370
2010-11-24	0.195110	0.151042	78.722928	0.040409	1.336813	4.364768	3.255555
...
2021-08-30	0.054910	0.053313	59.777959	0.025997	0.255526	0.398906	1.205450
2021-08-31	0.053420	0.056095	60.605257	0.025979	0.237575	0.401541	1.449005
2021-09-01	0.051732	0.058641	60.163852	0.025729	0.263501	0.514633	1.274227
2021-09-02	0.049573	0.058834	59.686587	0.024249	-0.020572	-0.166223	1.123236
2021-09-03	0.048928	0.057694	59.918896	0.023921	-0.028081	-0.081009	1.305459

2717 rows × 7 columns

Como ven definimos como indicadores o "features" 7 parámetros:

- 2 cruces de medias (pongo a propósito dos muy parecidos para mostrar algo luego)
- 1 indicador técnico (RSI)
- 1 oscilador sobre un indicador de volumen (OBV_osc)
- 3 indicadores paramétricos de la dist de retornos (sigma, skew y kurtosis)

Uso TSLA que no tiene dividendos y acoto con fecha de fin al 4-sept 2021 la serie para que puedan replicar en cualquier momento el mismo código y (si TSLA no cambia de política de dividend yield), debería coincidir la serie de datos en el futuro ya que la serie ajustada dará los mismos valores.

Vamos entonces a empezar a analizar los ítems que marcamos para comprender mejor la naturaleza de los indicadores que elegimos.

Correlación de los features con el target

Antes de ver esto, tenemos que definir el "target" es decir, no vamos a hacer una regresión, pero, si vamos a definir un método de trading, en función de estos indicadores, debemos saber contra qué operaremos, es decir compramos cuando los indicadores nos indiquen algo (luego veremos esos gatillos como los definimos) y la gracia entonces sería vender dentro de "X" velas.

El punto es definir ese "X" que en realidad muy posiblemente sea con un gatillo de venta por indicadores, pero inicialmente para hacer un estudio introspectivo de correlación de los indicadores vamos a suponer que se vende luego de una cantidad fija de velas, solo para estudiar la incidencia de los "features" o indicadores respecto al movimiento futuro del precio en esa cantidad "X" de velas

Calculamos el forward entonces (retorno futuro)

Uso 50 velas en el futuro porque si, fue arbitraria la elección (más adelante en la etapa de parametrización veremos como encontrar los valores óptimos de estos hiperparámetros), como usé indicadores que tenían un Rolling de 100 un RSI de 20 y cruces de 20 a 60 velas, me pareció razonable empezar viendo la incidencia de estos indicadores contra un forward de 50

```
data['fw'] = data.Close.shift(-100) / data.Close -1  
df = pd.concat([features, data['fw']], axis=1).dropna()  
df
```

Date	cruce1	cruce2	rsi	sigma	skew	kurtosis	OBV_osc	fw
2010-11-18	0.130694	0.100337	70.085913	0.045694	0.388374	4.494572	1.810285	-0.165942
2010-11-19	0.144293	0.109956	72.163427	0.045813	0.363155	4.405848	2.312055	-0.188771
2010-11-22	0.160633	0.122682	76.006381	0.045648	0.366442	4.417537	2.719116	-0.234132
2010-11-23	0.177235	0.137458	77.587571	0.043783	0.581596	4.828015	3.021370	-0.275962
2010-11-24	0.195110	0.151042	78.722928	0.040409	1.336813	4.364768	3.255555	-0.290668
...
2021-04-08	-0.104107	-0.120337	49.711164	0.044126	0.726778	2.478453	0.130817	0.068894
2021-04-09	-0.102155	-0.116309	48.910651	0.044115	0.725624	2.482173	0.033838	0.086703
2021-04-12	-0.098507	-0.109748	51.911527	0.044200	0.694266	2.411790	0.115405	0.045742
2021-04-13	-0.093130	-0.098065	58.164678	0.044903	0.660289	2.125191	0.259848	-0.039262
2021-04-14	-0.087435	-0.092148	54.448173	0.044499	0.701047	2.319189	0.047889	0.001830

2617 rows x 8 columns

Ahora que ya tengo contra que contrastar, voy a calcular la correlación entre mis features y ese FW

```
df.corr()["fw"]
```

	fw
crscl	0.135808
crscc	0.149471
rsi	0.132200
sigma	0.216300
scsi	-0.036702
varcasis	0.103399
SPX_DSE	0.189679
SPX_DSE	1.000000
fw	nan

names: fw, dtype: float64

Aprovechando la simpleza de pandas veamos el R² promedio de los features:

```
df.corr()["fw"][:-1].abs().mean()
```

```
0.13479428744700368
```

Como ven valores de R² en promedio en 0.134, y hago un paréntesis acá

No busquen R² de 0.9 acá, sencillamente porque eso no existe en finanzas, como dije páginas atrás la cantidad de ruido blanco en la bolsa es enorme, si bien hay pequeños sesgos estadísticos que podemos aprovechar (y para eso estamos acá) estos sesgos van a ser débiles, la gracia es ir aprovechando y juntando pequeños sesgos y buenas tácticas para construir estrategias sólidas pero ni sueñen con features en este tipo de bot que les den un R² con el rendimiento futuro de mas de 0.35 con toda la furia.

Multicolinealidad

Ovviamente a mas cantidad de features bien diferentes (repasar item construcción de indicadores de esta etapa de research) mayor probabilidad de construir una estrategia mas sólida.

Antes que se lo pregunten, si un indicador me da una pista del 10% de certeza ($R^2 = 0.1$), ¿es esperable que dos indicadores totalmente independientes de $R^2=0.1$ me den un 20%?

-No

Bueno, pero tampoco será del 10%, es decir a medida que sumo indicadores la certeza del conjunto de features combinados de forma óptima va a aumentar (no en forma lineal) pero va a aumentar claramente. En el tomo de inteligencia artificial vamos a ver varios modelos de ML para optimizar esa certeza de predicción a partir de nuestros features, por ahora nos concentraremos en el backtest de la estrategia en sí.

Entonces, ¿si voy juntando muchos, pero muchos indicadores siempre es mejor porque siempre suma mas certeza final al modelo o estrategia?

-Sí, pero..

Peeeeero, siempre hay un pero, en realidad no es cuestión de agregar indicadores porque si, dado que puede (y suele) pasar que entramos en multicolinealidad

¿Lo que?

En castellano, la multicolinealidad sucede cuando dos variables que están muy relacionadas entre sí. En este ejemplo puse a propósito dos cruces de medias muy parecidos, como vieron en los R^2 son cruce1=0.135 y cruce2=0.149 pero se superponen, es decir no me agrega nada de info nueva un cruce si ya tengo el otro, esto yo ya lo sé pero les voy a mostrar un método analítico para automatizar este tipo de filtro de multicolinealidad

Factor de inflación de la varianza

El FIV es una medida de la correlación de cada indicador con todo el resto, si el FIV de un indicador es bajo, significa que ese indicador no está ligado a ningún otro, o sea que su "efecto predictor" es totalmente independiente, en cambio si el FIV de un indicador es alto, me indica que ese indicador se superpone con otro, y por lo tanto entre todos los que se superponen deberá ir eliminando los de menor R^2 con el target.

Se calcula con la siguiente fórmula, super sencilla:

$$FIV_i = \frac{1}{1 - R_i^2}$$

donde FIV_i es el factor de inflación de varianza del indicador con cada "i" indicador restante de mis features, y R^2_i es el coeficiente de determinación entre el indicador y cada "i" indicador restante

Para simplificarnos la vida tenemos un método de statsmodels que lo calcula por nosotros, y como referencia, diremos que ese estadístico FIV debe ser menor a 10, para asegurarnos que no hay superposición de indicadores.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor  
  
X = df.iloc[:, :-1]  
cols = X.columns  
vif = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]  
  
pd.DataFrame(zip(cols, vif), columns=['feature', 'VIF']).set_index('feature')
```

feature	VIF
cruce1	32.163626
cruce2	32.056709
rsi	11.532822
sigma	11.580068
skew	1.240101
kurtosis	2.733694
OBV_osc	1.981997

Como ven, obviamente, como ya sabíamos, los dos cruces tienen alto FIV, y es porque ambos se correlacionan entre sí, si sacamos uno cualquiera, queda todo perfecto, en este caso conviene sacar el de menor R^2 de los dos, es decir el cruce 1 que tenía $R^2=0.135$ contra 0.149 del cruce2

Corroboramos esto que les dije, descartando el cruce1,

```
x = df.iloc[:, :-1].drop('cruce1', axis=1)
vif = [variance_inflation_factor(x.values, i) for i in range(len(x.columns))]
pd.DataFrame(zip(x.columns, vif), columns=['feature', 'VIF']).set_index('feature')
```

VIF

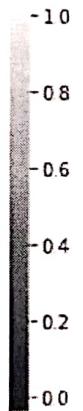
feature	VIF
cruce2	2.023174
rsi	10.882431
sigma	10.995432
skew	1.236685
kurtosis	2.729176
OBV_osc	1.970381

Ahora me quedan el rsi y el sigma que están ahí en el palo, pero podemos dejarlos, claramente los que se superponían eran los dos cruces como sabíamos.

A todo esto, lo del factor de inflación de varianza es la sensibilidad del aumento de varianza (de ahí el nombre) asumiendo una regresión lineal, y no es nuestro caso, nosotros solo queremos ver correlación o mejor dicho solo nos interesa saber el "nivel de predictibilidad" del cada indicador con el target, forward o retorno futuro. Pero de todos modos si bien no vamos a hacer ninguna regresión, nos sirve para descartar indicadores superpuestos

Otra forma de ver que indicadores se correlacionan mucho entre sí es usando un heatmap de correlación, en este caso vemos como los cruces claramente se superponen con un R^2 casi de 1

```
import seaborn as sns
sns.heatmap(df.corr(), annot=True, fmt='.%', linewidths=1)
```



Autoregresión de Indicadores

Muchas veces tenemos indicadores autoregresivos, es decir indicadores que correlacionan muy fuerte consigo mismos en una cantidad de velas de diferencia. Esto en sí no es necesariamente un problema, puede serlo si tiramos los features así nomás en un modelo de machine learning y luego no somos capaces de interpretar los resultados del modelo, pero por ahora que solo nos estamos enfocando en conocer los features, vamos a analizar en que me jode que un indicador sea autoregresivo

Para ello primero tengo que entender que es un indicador autoregresivo, y como dije antes es un indicador que correlaciona consigo mismo "x" velas hacia atrás, por ejemplo un cruce de medias 50/200, es obvio que el valor de hoy va a estar muy ligado al de ayer ¿Por qué? Sencillamente porque solo hay un día (una vela) de diferencia en un indicador que pondera linealmente 200 velas, es decir que la incidencia de un solo día es muy baja, con lo cual es obvio que sabiendo el cruce de hoy puedo saber casi con precisión el valor del cruce de mañana.. Pero que pasa con el cruce 5/10, y si lo comparo con 20 velas adelante, ahí ya pierdo toda la data pasada de arrastre, se supone que no correlaciona... bueno, no se supone nada acá, si hay algún tipo de estacionalidad va a haber igual una autocorrelación

Ahora se preguntarán ¿Cómo demonios evalúo esto?

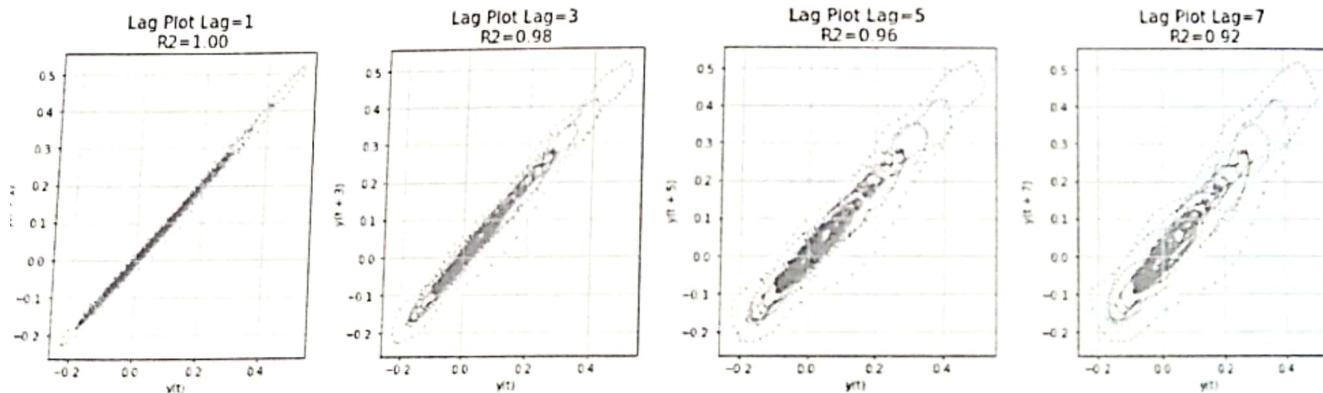
Bueno, se usan gráficos de LAG

Vemos nuestro ejemplo con el cruce1

```
fig, ax = plt.subplots(figsize=(20,5), ncols=4)

for i, lag in enumerate(range(1,9,2)):
    pd.plotting.lag_plot(df['cruce1'], lag=lag, s=1, ax=ax[i])
    ax[i].grid()
    df1 = df['cruce1'].iloc[:~-lag].reset_index(drop=True)
    df2 = df['cruce1'].iloc[lag: ].reset_index(drop=True)
    r2 = df1.corr(df2)
    ax[i].set_title(f'Lag Plot Lag={lag} \nR2={r2:.2f}', fontsize=15)

plt.subplots_adjust(wspace=0.3)
```



¿Y esto?

Bueno, es bastante obvio, con un lar de 1, en un cruce 20/60, es obvio que la correlación es casi perfecta de $R^2=1$, pero a medida que voy aumentando el lag ese R^2 se va desvaneciendo, hasta que obviamente pasado el lag=60 pierdo el arrastre estadístico y debería tener un R^2 insignificante, a menos que el indicador presente cierta estacionalidad cada "n" velas..

Pero claro, para verificar esto tendría que hacer muchos gráficos de LAG con valores incrementales de Lag hasta infinito o hasta la duración de la serie.. bueno, no, por suerte hay otro tipo de gráfica que son justamente gráficas de autocorrelación que hacen todo ese trabajo de una y lo podemos hacer de una también en un frame para todos los indicadores... que simple es la vida con Python eh!

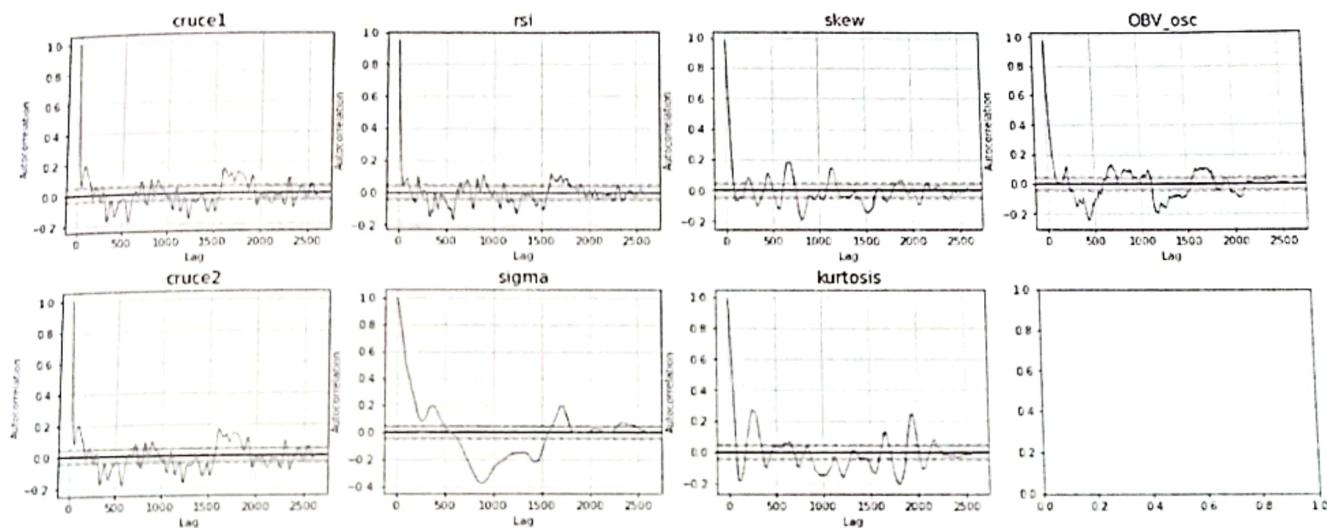
Veamos el código:

```
n_rows = 2
n_cols = len(df.columns)//2

fig, ax = plt.subplots(figsize=(20,8), ncols=n_cols, nrows=n_rows)

for i, col in enumerate(df.columns[:-1]):
    r = i%2
    c = i//2
    pd.plotting.autocorrelation_plot(df[col], ax=ax[r][c])
    ax[r][c].set_title(col)

plt.subplots_adjust(hspace=0.3)
```



Ahora a interpretar esto

Tomemos el del cruce1, como habíamos dicho en la página anterior, para un lag de 1 el $R^2=1$ y va bajando el R^2 (eje Y) a medida que aumenta el lag (eje X) hasta que pierde el arrastre estadístico y se mete en una zona de insignificancia (demarcada por la franja punteada)

Pero vemos que por momentos se sale de la zona de insignificancia.. Bueno, eso es porque hay cierta estacionalidad, es decir como que en una cantidad de velas aparece como un patrón de correlación una vez perdido todo el arrastre estadístico ¿Qué significa esto?

Para analizarlo tomemos el sigma, fíjense que mas o menos para un lag=900 aparece un R^2 de -0.4. Tenemos 2600 datos de TSLA, es decir que en ese lag=900 tenemos 3 períodos completos, o sea que cada 900 velas, mas o menos 3 años, hubo una correlación inversa fuerte, es decir, si un día sigma tenía un valor bajo, 900 velas adelante tenía uno alto (con un $R^2=0.4$, es un comportamiento estacional medio raro, pero son solo 3 períodos completos, es decir en esos valores de lag tan altos respecto al total de datos, no vamos a tomar muy en serio "lo estacional" del indicador.

Ahora si se fijan el OBV_osc, que tiene un arrastre estadístico de 100 velas, se sale de la banda en lag=400 aprox, superando el -0.2.. y acá con lag=400 ya tenemos 7 períodos enteros dentro de nuestra serie, ya empezamos a darle un poco más de bola, esto se pudo deber a ciclos de flujo de volumen de capital, y si se fijan la kurtosis, con un arrastre estadístico de 100 velas, se sale de la banda en un lag=250 aprox más o menos cada año, esto se pone interesante, porque tenemos 10 períodos completos donde se presenta una estacionalidad de la kurtosis incluso más alta que el R2 de la kurtosis contra nuestro target de forward, esto quiere decir que este indicador es más autoexplicado que predictor.

Esto de un indicador paramétrico que se explique a si mismo (con cierta correlación y estacionalidad sabidas) puede ser una gran ventaja para operar por ejemplo opciones en donde los parámetros de la distribución son muy útiles por ejemplo para operar volatilidad, pero no quiero hacer un cholo acá con eso, ya habrá un tomo entero dedicado a opciones, solo lo menciono porque tenía el gráfico con ese detalle a mano, la idea acá en la introspección de los indicadores es que ustedes vayan conociendo mejor la naturaleza de lo que están usando en sus modelos de trading.

Distribuciones de Features y Target

Esto es crucial, es decir, es muy importante saber si los valores de mis indicadores o mi target (forward de rendimiento) se distribuyen uniformemente entre dos extremos (distribución uniforme) o si siguen una distribución normal o si siguen otro tipo de distribución (repasar t[6] distribuciones de variable continua en los retornos de activos de la bolsa).

Iba a explayarme pero realmente en el tomo 6 está muy bien explicado los parámetros de cada distribución y como realizar test de hipótesis con scipy como el KS o cualquier otro para fittear las distribuciones y ver si corresponden a normales o a otro tipo de distribución y en tal caso parametrizarlas para modelar simulaciones luego, así que no voy a volver a mostrarlo.

Balanceo o Desbalanceo

Generalmente los retornos en la bolsa tienen un balanceo muy cercano al 50%-50% es decir la mitad de las velas sube y la otra mitad baja, lo mismo un oscilador como el RSI tendrá la mitad de los valores por debajo y la otra mitad por encima del 50, y así, ahora hay casos en los que buscamos algo muy desbalanceado, esto es típico en los bots que operan contrarian, buscan valores fuera de los 2 o mas sigmas de su media, con lo cual serán eventos de probabilidades del orden del 1% o menores, este tipo de operativa es muy útil combinada con un buen screener, ¿Por qué? Y porque si estoy esperando que se dé una situación determinada que solo pasa el 1% de las veces, puedo estar clavado en la pantalla días hasta que se dé el evento, pero si tengo un screener que minuto a minuto revisa los 10.000 activos de USA, voy a encontrar todos los minutos, unos 100 activos que tienen en ese momento un evento con probabilidad 1% de ocurrencia

A lo que voy con este punto es a lo siguiente.

Hasta ahora yo les mostré correlación o buscar el R^2 de un feature contra un target, por ejemplo buscamos la correlación de un cruce de medias contra el fw o P&L a XX velas adelante, ahora bien, ese R^2 es de todos los percentiles del cruce, cuando a lo mejor me interesa solo los percentiles mas extremos, veamos un ejemplo.

Vamos a tomar como feature un cruce de medias cualquiera, el 10-30, y vamos a ver como correlaciona con un fw de 10 velas, como es de esperar la correlación será débil en toda la serie, pero vean que interesante se pone si tomamos no se por ejemplo el primer decil o 10% superior o inferior.

```
import yfinance as yf
import numpy as np

data = yf.download('TSLA', auto_adjust=True, end='2021-09-04', progress=False)
data['cruce'] = data.Close.rolling(10).mean() / data.Close.rolling(30).mean() -1
data['fw'] = np.log(data.Close.shift(-10) / data.Close)
data = data.dropna()
data
```

Date	Open	High	Low	Close	Volume	cruce	fw
2010-08-10	3.930000	3.930000	3.764000	3.805000	6406500	0.016948	0.008894
2010-08-11	3.738000	3.776000	3.570000	3.580000	3988000	0.012969	0.105919
2010-08-12	3.560000	3.580000	3.478000	3.520000	3455000	0.009677	0.115255
2010-08-13	3.636000	3.690000	3.532000	3.664000	3170000	0.007653	0.072625
2010-08-16	3.690000	3.760000	3.652000	3.756000	2429000	-0.002598	0.056419
...
2021-08-16	705.070007	709.500000	676.400024	686.169983	22677400	0.048724	0.063165
2021-08-17	672.659973	674.580017	648.840027	665.710022	23721300	0.041898	0.099995
2021-08-18	669.750000	695.770020	669.349976	688.989990	20349400	0.036391	0.063405
2021-08-19	678.210022	686.549988	667.590027	673.469971	14313500	0.029273	0.083870
2021-08-20	682.849976	692.130005	673.700012	680.260010	14781800	0.025323	0.075448

2778 rows x 7 columns

Como dijimos, la correlación es débil en general

```
data.cruce.corr(data.fw)

0.04199618520204466
```

Pero vean que interesante se pone en el primer y último 10% de datos de cruces

```
primer_decil = data.sort_values('cruce', ascending=True).iloc[:256]
primer_decil.cruce.corr(primer_decil.fw)

-0.4600510120664438

ultimo_decil = data.sort_values('cruce', ascending=False).iloc[:256]
ultimo_decil.cruce.corr(ultimo_decil.fw)

-0.16662526884350282
```

O sea, en lugar de un $R^2 = 0.04199$ que es muy débil, obtenemos:

- $R^2 = -0.46$ Para los cruces más bajos (primer decil, primeros 256 datos de 2557)
- $R^2 = -0.166$ Para los cruces más bajos (último decil, últimos 256 datos de 2557)

O sea que tanto para valores extremos de cruces positivos como negativos, la correlación es negativa, ojo hay que interpretar esto correctamente porque es medio confuso, que el R^2 sea negativo significa que a mas de la variable X menos de la Y y viceversa.

Pero lo que nos importa no es la proyección, recuerden, sino que haya una correlación y que los deciles me sirvan para salir del sesgo del 50%-50%, así que veamos lo que dan las medias o medianas en estos deciles de cruces:

```
primer_decil.fw.mean(), ultimo_decil.fw.mean()  
(0.028581496811011384, 0.049089030864067455)  
  
primer_decil.fw.median(), ultimo_decil.fw.median()  
(0.022045038691176508, 0.036448673879078096)
```

Si bien ambos me dan positivos, el ultimo decil si bien el R^2 era más débil el sesgo es mayor ya que la media y mediana me dan mas que en el primer decil. O sea que es mucho mejor indicador este del cruce de 10/30 para un screener que busque valores extremos de este cruce que como indicador en general. Esto es algo muy común de ver en diferentes indicadores que en valores extremos suele ser mejores predictores, así que no quería dejar de poner este apartado para que busquen no solo features en general, sino que usen gatillos de features en zonas mas extremas o al menos lo tengan en cuenta.

Veámoslo en algunos ratios, si bien no me quiero adelantar con los ratios para medir performance y riesgo, al menos el simple ratio de suma de FW ganadores vs suma de FW perdedores, llamado "profit factor" que mide la asimetría del área positiva de la distribución respecto de la negativa, si el ratio es mayor a 1 en retornos logarítmicos, es un sesgo positivo, vemos que hay una marcada mejora en el ratio usando los FW ganadores y perdedores del ultimo decil, es decir, ese último decil nos genera resultados mas diferenciados

```
# Separamos los FW Positivos de Los Negativos del ultimo Decil  
  
ganadores = ultimo_decil.fw.loc[ultimo_decil.fw > 0]  
perdedores = ultimo_decil.fw.loc[ultimo_decil.fw < 0]  
  
# Profit Factor  
profit_factor = -ganadores.sum() / perdedores.sum()  
profit_factor  
  
2.530105752433885  
  
# Profit Factor Dist Original  
-data.fw.loc[data.fw>0].sum() / data.fw.loc[data.fw<0].sum()  
1.6017677902322223
```

Otro aspecto que se suele usar mucho cuando buscamos métricas desbalanceadas o deciles o percentiles extremos de un determinado indicador es el ratio de colas, es decir lo mismo pero en lugar de usar toda la dist, usamos solo las colas o quantiles 0.05 y 0.95, a este ratio de colas se lo llama Rachev Ratio.

```
# Rachev Ratio
rr = ganadores.quantile(0.95) / abs(perdedores.quantile(0.05))
rr
1.9773065811125419

data.fw.quantile(0.95) / abs(data.fw.quantile(0.05))
1.359281899137708
```

Vemos nuevamente como el ultimo decil tiene un ratio más marcado.

Planteo de "Racional" o Hipótesis de trabajo.

Bueno, una vez que tenemos resuelto todo el tema de data-feed, y que ya investigamos bien las correlaciones y armamos una buena lista de indicadores o features para usar en el sistema como gatillos tanto de compra como de venta, vamos a delinejar el esquema del modelo de bot que queremos backtestear

Esto ya lo dijimos, pero repasando lo que tenemos que definir son las siguientes cuestiones:

- 1- ¿Qué tipo de Bot será?
- 2- Definir si es backtesteable y tipo y alcance de backtest a realizar
- 3- Definir Activos, Mercados, Horarios, Timeframes, Features, fuentes de datos
- 4- Inputs Parametrizables (variables)
- 5- Definir flujo de gatillos y acciones del modelo a backtestear
- 6- Objetivos del Bot

Es muy importante que no haya ambigüedades tipo "va a operar por cruce de medias" o sea, tiene que quedar bien claro que medias ¿simples, exponenciales, triangulares? ¿De qué ventana de velas, es parametrizable este dato o es fijo? ¿Qué timeframe? ¿Se considera cruce cuando la rápida es mayor que la lenta o se espera un valor x% mayor configurando ese "x" como parametrizable? En fin, como ven tiene que quedar super claro y objetivo la redacción de este racional porque en función de ello viene todo el resto del desarrollo

Dicho esto, les voy a poner un ejemplo para bajar a tierra un poco todo, insisto que los ejemplos son meramente a modo didáctico, la idea es que cada uno a donde en cada punto por separado como fuimos viendo, es de esperar que un buen modelo tenga al menos unos 10 features bien diferenciados uno de otros.

Ejemplo de Racional

- 1- Bot de Swing trading tipo Trend-Following
- 2- Es Backtestable. Mediaremos la performance y ratios de riesgo 20 años hacia atrás mediante un backtest matricial de los potenciales trades que debieron haberse realizado de seguir el racional, asumiendo que los volúmenes operados no afectan al mercado ni al precio del activo a operar
- 3- Inputs:
 - Activos: Inicialmente probamos con TSLA
Apuntamos a escalar aEquity USA con mktCap > 10 B (ver portabilidad mas allá)
 - Mercados: Solo USA NYSE/NASDAQ/CBOE
 - Se usará un timeframe diario para el backtest
 - Se operará sobre los últimos 5 minutos de rueda asumiendo operaciones a precios de cierre
 - Usaremos como fuente de datos yfinance
 - Usaremos como features:
 - o Volatilidad (desvío estandar de los retornos diarios de "n" velas parametrizable)
 - o Cruce de medias simples tipo 20/60 (parametrizables los valores)
 - o RSI, con una ventana de "n" velas parametrizable
 - o Oscilador sobre el OBV con una ventana "n" velas parametrizable
- 4- Parametrizables:
 - o Valores para el cruce de medias (rápida y lenta)
 - o n_RSI (la ventana para construir el indicador)
 - o n_desvio_estandar (la ventana de retornos back para medir el desvío)
 - o n_obv (la ventana para construir el oscilador)
 - o Gatillos:
 - El % necesario del cruce tanto en compra como en venta
 - El valor de RSI, Volatilidad y OBV para disparar gatillo compra/venta
- 5- Flujo del BOT:
 - 1- El bot espera que se de gatillo de compra con las siguientes señales (a parametrizar)
 - o El cruce de la media de 20 velas con la de 60 velas es positivo
 - o El RSI (ventana de 15 velas) es mayor a 60
 - o La volatilidad (ventana de 40 velas) es mayor al 1% diaria
 - 2- Queda comprado hasta que se de gatillo de venta
 - 3- Vende cuando se cumple el siguiente gatillo:
 - Cruce de medias 20/60 es menor a -1 %
 - El RSI (ventana de 15 velas) menor a 55
 - EL OBV oscilador construido con ventana de 100 velas es > 0
 - 4- No hay ningún tipo de stopLoss ni TakeProfit ni estático ni dinámico
 - 5- No hay ningún tipo de manejo de capital asociados a los trades, todos los trades son por un mismo monto genérico, simplemente accionados por gatillos descriptos
 - 6- Objetivo del Bot: Al ser una idea de swing y tren-following la idea es pasar el trilema de los bots de profit en trading algorítmico (a continuación, explico esto) buscando recortar riesgo y ser rentable en el largo plazo (aunque quizás no logre superar al benchmark)

PD: Típico objetivo de un bot de swing del tipo tren-following.

Va primera aclaración:

El flujo descripto es absurdamente sencillo y solo a fines didácticos, en la realidad se usan gatillos de muchos más indicadores, se usan modelos de stop los dinámicos y lo mismo para take profit, y se usa un manejo de capital y volumen por trade, lo que pasa es que para bajar el código a algo reproducible en un libro y con fines didácticos hay que redondear los ejemplos a algo sencillo de explicar paso a paso, en la práctica la idea es que a ustedes les sirva el framework de trabajo para backtestear justamente sus ideas paso a paso de un modo ordenado.

El trilema de los bots de Profit

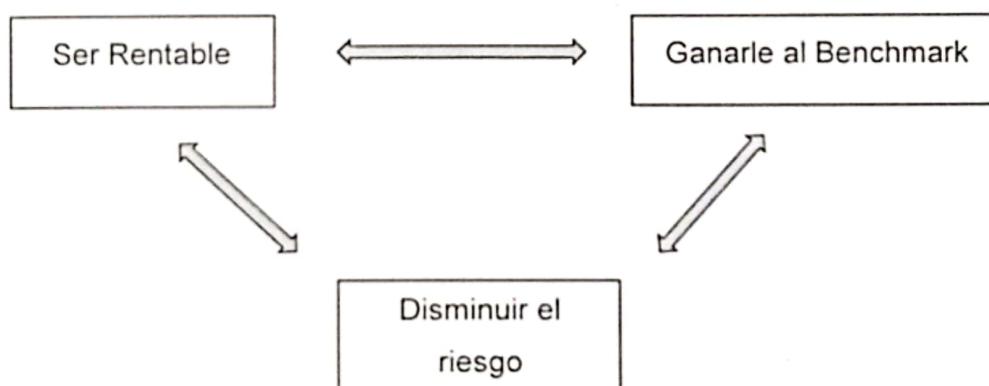
Pongo este tema aparte porque merece que le prestemos atención, aclaro que esta idea que tuve de hablar del "trilema" es una analogía a otros trilemas muy conocidos, ustedes saben a que me refiero, la idea es que en primer lugar cumpla al menos 2 de los siguientes objetivos:

- 1- Sea rentable (CAGR positivo)
- 2- Le gane al Buy&Hold (u otro benchmark)
- 3- Disminuya riesgo respecto al Buy&Hold

¿Por qué 2 de 3?

Sencillo, porque es imposible (?) asegurar cumplir los 3 objetivos en la práctica

No digo que no se pueda dar, sino que se puede dar de vez en cuando en situaciones especiales, pero la realidad es que, de forma sistemática se puede apuntar perfectamente a cumplir 2 de 3, pero se hace casi imposible intentar cumplir los 3 objetivos con el mismo BOT



Ejemplo, si quiero ganarle al benchmark (ej SP500) tengo que asumir mas riesgo, o si quiero disminuir el riesgo debo resignar rentabilidad.

Ejemplo, hay métodos defensivos que cortan rápido pérdidas, y le ganan al benchmark claramente en las crisis, y a su vez disminuyen el riesgo, pero en los bull markets no pueden superar al benchmark.

En fin, como les decía, generalmente se puede lograr 2 de 3 objetivos del trilema, pero se torna imposible si queremos los 3 al mismo tiempo con el mismo método

¿La solución al trilema?

Si la supiera no estaría acá escribiéndolo en un libro seguramente, pero mas allá del chiste, asumiendo que todos somos igual de mortales y falibles en el mercado, creo que lo más razonable es tener un bot bueno en cada objetivo, e ir combinando todos en función de los objetivos personales y visión del mercado, por ejemplo cuando estamos mas temerosos y necesitamos mas seguridad porque vemos mucha volatilidad en el mercado, asignarle mas capital a un bot mas defensivo cuyo principal objetivo sea bajar el riesgo de exposición al mercado.

Simulación de Gatillos de Compras y Ventas

Bien ahora ya se acaba el palabrerío y vamos a empezar a codear y probar todo esto que venimos diciendo y a backtestear de una vez el modelo. Lo primero que vamos a construir son los gatillos de compra y venta

Para armar los gatillos de compra y venta obviamente partimos del racional que definimos en el punto anterior, primero calculamos los "features" como dije, es un modelo hiper burdo, ya vimos mas en detalle como construir indicadores mas interesantes y como evaluar la correlación contra un forward, ahora nos concentraremos en, dados los features, evaluar el backtest.

```
import yfinance as yf, numpy as np, pandas as pd

# Hiperparametro => para el oscilador OBV
n_obv = 100

# Hiperparametro => para el sigma
n_sigma = 40

# Hiperparametro => para el RSI
n_rsi = 15

# Hiperparametro => para el cruce
fast, slow = 20, 60

data = yf.download('TSLA', auto_adjust=True, progress=False, start='2011-01-01', end='2020-12-31')
data['Balance'] = np.where(data.Close>data.Close.shift(), data['Volume'],
                           np.where(data.Close < data.Close.shift(), -data['Volume'], 0))

data['OBV'] = data['Balance'].cumsum()
dif = data['Close'].diff()
win = pd.DataFrame(np.where(dif > 0, dif, 0), index=data.index)
loss = pd.DataFrame(np.where(dif < 0, abs(dif), 0), index=data.index)
ema_win = win.ewm(alpha=1/n_rsi).mean()
ema_loss = loss.ewm(alpha=1/n_rsi).mean()
rs = ema_win / ema_loss

# Features
data['cruce'] = data.Close.rolling(fast).mean() / data.Close.rolling(slow).mean() -1
data['rsi'] = 100 - (100 / (1+rs))
data['sigma'] = data.Close.pct_change().rolling(n_sigma).std()
data['OBV_osc'] = (data.OBV - data.OBV.rolling(n_obv).mean()) / (data.OBV.rolling(n_obv).std())
features = data.iloc[:, -4:].dropna()
features
```

La salida del script de la página anterior es el siguiente dataframe de "features", uso una fecha de inicio y fin fija 2011-2020 para que ustedes puedan reproducir exactamente el mismo backtest.

Date	cruce	rsi	sigma	OBV_osc
2011-05-25	0.066759	60.657683	0.039441	2.154925
2011-05-26	0.066522	62.525004	0.039414	2.737019
2011-05-27	0.066726	62.700011	0.029056	2.922602
2011-05-31	0.068359	65.023007	0.028402	3.334472
2011-06-01	0.069083	55.266080	0.029319	2.884899
...
2020-12-23	0.247179	60.972635	0.041372	1.655826
2020-12-24	0.246201	63.025740	0.040430	1.692907
2020-12-28	0.245130	63.277422	0.040462	1.757391
2020-12-29	0.244565	63.595475	0.038955	1.705833
2020-12-30	0.244407	67.384055	0.039124	1.892423

2417 rows x 4 columns

Una vez armada la matriz de features, se procede a armar una matriz de "gatillos"

Hay mil formas de armar esto, para mi, lo mas ordenado es usando "máscaras"

La máscara no es ni mas ni menos que una condición pero asociada a una columna o varias al mismo tiempo, en nuestro caso, tenemos 3 condiciones a verificar como gatillo de compra:

- Cruce de Medias (20/60) Positivo
- RSI (ventana de 15 velas) > 65
- Volatilidad diaria (ventana 40 velas) > 1%

Lo que hacemos entonces es asignar un valor de True/False a la columna de cada condición del gatillo y luego definimos una máscara que pregunte si TODOS por eso el "all()" los valores de las COLUMNAS por eso el "axis=1" son verdaderos, de ese modo la máscara será verdadera, si no, será falsa, como vemos, al evaluar el True como 1, la suma me da 460 de 2516 valores cuyas 3 columnas era True.

```
gatillos_compra = pd.DataFrame(index = features.index)
gatillos_compra['cruce'] = np.where(features.cruce > 0, True, False)
gatillos_compra['rsi'] = np.where(features.rsi > 65, True, False)
gatillos_compra['sigma'] = np.where(features.sigma > 0.01, True, False)
mascara_compra = gatillos_compra.all(axis=1)
mascara_compra.sum(), mascara_compra.count()

(460, 2417)
```

Pero si quieren pueden inspeccionar uno por uno, o al menos unos 10 o 20 valores para chequearlo, siempre recomiendo hacer este tipo de cosas, por mas confiados que estemos, a veces, nos comemos una coma o algo simple en el código y el resultado es totalmente diferente a lo que pensamos, así que allí va el mini-debug de esto, para ello imprimimos el dataframe de los features, luego el dataframe de los "gatillos_compra" para ver uno por uno los valores de verdad de cada gatillo, y luego la máscara

Entonces recordemos las condiciones:

- Cruce de Medias > 0
- RSI > 65
- Volatilidad, sigma > 0.01

Y ahí evalúan cualquier valor, por ejemplo el del 26 de mayo, el cruce 0.06 es mayor a cero por lo que tenemos un True en su columna de gatillos, luego el valor de RSI está en 62.52 que es menor que 65 que era la condición a superar, por lo que el gatillo en RSI esta como False, y finalmente el del sigma dio 0.0394 y debería superar el 0.01 con lo cual otro True, pero para esa fecha como no son los 3 True (all) ya que uno dio False, la máscara da False para esa fecha como gatillo (ya que deben cumplirse las tres condiciones juntas)

features	gatillos_compra			mascara_compra
Date	cruce	rsi	sigma	
2011-05-25	0.066759	60.657683	0.039441	2.154925
2011-05-26	0.066522	62.525084	0.039414	2.737019
2011-05-27	0.066726	62.790011	0.029056	2.922802
2011-05-31	0.068359	65.023007	0.028402	3.334472
2011-06-01	0.069083	55.266080	0.029319	2.884699
...
2020-12-23	0.247179	60.972635	0.041372	1.655826
2020-12-24	0.246281	63.025740	0.040430	1.692507
2020-12-28	0.245130	63.277422	0.040462	1.757391
2020-12-29	0.244565	63.595475	0.038955	1.795833
2020-12-30	0.244407	67.384055	0.039124	1.892423
2417 rows × 4 columns		2417 rows × 3 columns		
		2417 rows × 1 columns		

Este tipo de debug, lo recomiendo siempre, pero obviamente no es necesario, es simplemente para chequear que estamos bien encaminados y el código realmente es fiel a la idea original de gatillos

Lo mismo hacemos para el gatillo de venta, tenemos que generar una máscara que nos indique cuales son las fechas que cumplieron todas las condiciones de venta prefijadas:

- Cruce de Medias < -1%
- RSI < 55
- Oscilador OBV > 0

```

gatillos_venta = pd.DataFrame(index = features.index)
gatillos_venta['cruce'] = np.where(features.cruce < -0.01, True, False)
gatillos_venta['rsi'] = np.where(features.rsi < 55, True, False)
gatillos_venta['obv'] = np.where(features.OBV_osc > 0, True, False)
mascara_venta = gatillos_venta.all(axis=1)
mascara_venta.sum(), mascara_venta.count()

(113, 2417)

```

Como vemos el gatillo de venta es mas estricto, o sea, las condiciones son mas difíciles de cumplir todas simultaneas. Esta métrica va a resultar muy interesante luego al hacer un análisis de sensibilidad de parámetros.

Una vez que tenemos la simulación de cuando se activan los gatillos bajada a una máscara de compra y venta, debemos armar la tabla de Acciones

Armado de tabla de Acciones

Probablemente te preguntes ¿Qué pasa si hay un gatillo de compra dos días seguidos?

Es clave entender que el gatillo es único, es decir si el día 1 da compra, y el día 2 también, la compra se realizaría el día 1 y no debería repetirse el día 2, pero nuestra máscara de gatillos contempla los 2417 días con datos completos de features, y de ahí necesitamos pasar a una matriz de acciones que contemple solo cuando hay un "cambio de estado" es decir de "gatillo de compra" a "gatillo de venta" o viceversa, ya que hay situaciones como por ejemplo:

- *3 días seguidos de gatillo de compra: En este caso, solo el primer día nos interesa como acción de compra*
- *1 día de gatillo de compra, 5 días sin nada, 2 días de gatillo de compra, y un 10º día con el gatillo de venta: En este caso, nos interesan solo las filas del gatillo de compra inicial y el de venta final*

Son solo ejemplos al azar que tiro, la idea es que me quede una tabla de acciones final con compra, venta, compra, venta, y así...

Para ello podemos usar un iterador (mala idea) o podemos seguir operando con filtros y matrices aprovechando las bondades de pandas que es lo que voy a hacer:

```
data.dropna(inplace=True)
data['gatillo'] = np.where(mascara_compra, 'compra', np.where(mascara_venta, 'venta', ''))  
actions = data.loc[data.gatillo != ''].copy()  
actions['gatillo'] = np.where(actions.gatillo != actions.gatillo.shift(), actions.gatillo, '')  
actions = actions.loc[actions.gatillo != ''].copy()  
actions
```

El output de esto se los pego en la siguiente página que acá no me entra, pero analicemos mientras tanto el código, parece medio rebuscado pero es super eficiente en resolver el filtrado de los días de acciones solamente sin tener que iterar (por si se preguntan por qué esquivar la iteración recorriendo uno por uno los valores de la matriz, es porque ese es un proceso muy lento, mientras que el proceso de filtros matriciales es infinitamente más eficiente en términos de velocidad)

- 1- Les decía primera linea luego de dropear los datos vacíos, armo una columna en "data" con la info de si ese día se dispara o no el gatillo de compra, venta o ninguno (al ninguno lo dejo en blanco)
- 2- Luego armo una nueva matriz de acciones "actions" como copia de la anterior, pero solo elijo los días que hay señal de algo (compra/venta) limpio los días sin señal de ningún tipo
- 3- Tercer paso, si hay cambio de estado, es decir si el gatillo es distinto al anterior (compra y luego venta o al revés) le dejo el valor original, caso contrario (gatillo repetido) lo borro o dejo en blanco
- 4- Cuarto paso, dropeo, los datos en blanco (gatillos repetidos del paso anterior)

Veamos cómo queda la tabla final actions:

Date	Open	High	Low	Close	Volume	Balance	OBV	cruce	rsi	sigma	OBV_osc	Gatillo
2011-05-31	5 938000	6 056000	5 910000	6 026000	16452500	16452500	118666000	0 068359	65 023007	0 028402	3 334472	compra
2012-01-18	5 338000	5 376000	5 250000	5 362000	6301000	6301000	80177500	-0 083017	44 723758	0 047041	0 003065	venta
2012-02-15	6 620000	6 682000	6 454000	6 720000	13809000	13809000	127319000	0 009988	66 937196	0 044627	2 792804	compra
2013-11-11	28 200001	29 094000	27 420000	28 940001	69988000	69988000	1228801000	-0 021696	37 997823	0 041509	0 003973	venta
2014-01-16	32 500000	34 540001	32 480000	34 194000	59797000	59797000	1481964000	0 028473	66 157930	0 044273	1 297222	compra
2014-04-22	41 271999	43 866001	41 001999	43 728001	49023500	49023500	1536108000	-0 017390	53 431157	0 037299	0 163592	venta
2014-06-16	41 352001	45 098000	41 251999	44 922001	66232000	66232000	1525634000	0 000109	66 602118	0 032350	-0 343382	compra
2014-10-15	44 000000	46 198002	43 464001	45 939999	45736500	45736500	1669980500	-0 021526	35 642476	0 027298	0 073664	venta
2015-04-24	44 099998	44 160000	43 602001	43 686001	12139000	-12139000	1398322000	0 003622	65 505542	0 021731	-0 452249	compra
2015-08-18	51 076000	52 189999	50 712002	52 144001	20975000	20975000	1484308500	-0 010631	53 378474	0 028123	0 730114	venta
2016-03-17	44 293999	45 700001	44 000000	45 276001	18914500	18914500	1474843000	0 002427	72 385048	0 035815	2 566702	compra
2016-05-20	43 397999	44 110001	43 270000	44 056000	45035500	45035500	1577211000	-0 016339	47 112499	0 022806	1 010767	venta
2016-12-23	41 599998	42 689999	41 542000	42 667999	23352500	23352500	1372508500	0 000129	69 928612	0 020594	0 069821	compra
2017-07-26	68 071999	69 099998	67 624001	68 769997	24104000	24104000	2097021500	-0 011907	53 076252	0 025141	1 352485	venta
2017-09-15	74 902000	76 000000	74 540001	75 961998	27102500	27102500	2205684500	0 018672	65 133157	0 022764	1 429404	compra
2017-10-24	67 760002	68 559998	67 232002	67 468002	22458500	22458500	2183333000	-0 011573	40 209417	0 019794	0 945891	venta
2018-01-22	69 879997	71 506002	69 839996	70 311995	31052000	31052000	2125269500	0 026197	65 322862	0 019258	0 216856	compra
2019-01-14	68 475998	68 500000	65 800003	66 879997	26236500	-26236500	1593132000	-0 013066	50 083332	0 036498	0 368072	venta
2019-07-12	47 950001	49 076000	47 942001	49 015999	46002500	46002500	699972000	0 009579	66 579924	0 029799	-0 782325	compra
2020-03-23	86 720001	88 400002	82 099998	86 858002	82272500	82272500	2388793000	-0 021433	34 312457	0 085488	0 584441	venta
2020-06-01	171 600006	179 800003	170 820007	179 619995	74697500	74697500	3244588000	0 209053	68 971247	0 046852	1 166281	compra
2020-11-16	408 929993	412 450012	404 089996	408 089995	26838600	-26838600	4464774200	-0 012067	46 553691	0 034105	0 834046	venta
2020-11-20	497 989990	502 500000	489 059998	489 609985	32911900	-32911900	4633569900	0 005556	65 390005	0 034293	1 176707	compra

Como ven, todo se redujo solamente a los días que hay cambio de estado por pasar de un gatillo a otro, obviamente pueden seguir "debugueando" verificando celda a celda que este todo correcto es decir que cada gatillo realmente cumpla las condiciones que debía cumplir, vean por ejemplo a vistazo rápido que los cruces pasan siempre de positivos a negativos para gatillos de compra y venta respectivamente, el RSI va de >65 a <55 celda a celda, y así.

A todo esto, la idea es que vayan siguiendo el código paso a paso para que vayan probando y debugueando línea por línea y además la idea es que vayan probando jugar con algunas variables o parámetros propios, como les dije de antemano esto es solo un modelo de apoyo para mostrar el "pipeline" o esquema de trabajo paso a paso, pero claro está después de todos los indicadores que vimos, que es muy pobre esto así como lo estamos armando con estos features.

Bueno, sigamos, no sé si lo notaron, pero la table tiene un problema

"Empieza en compra y termina en compra"

Esto por suerte es bastante fácil de solucionar, tendremos que primero asumir si:

- El método es siempre LONG, es decir arranca comprando y termina vendiendo
- Es siempre SHORT, o sea que arrancaría vendiendo y terminaría recomprando,
- Es mixto, es decir que abre posición LONG comprando, cierra vendiendo, pero a su vez en ese mismo instante abre la SHORT, que luego cierra recomprando al mismo instante que vuelve a abrir la LONG y así

En nuestro caso por simplicidad arrancamos suponiendo que es siempre LONG
 Es decir que la primera fila tiene que ser una compra y la última una venta.

Para ello ponemos simplemente unos IF y sliceamos y sacamos lo que no corresponda:

```
# Para métodos solo LONG debo descartar el primero si es venta y el ultimo si es compra
if actions.iloc[0].loc['gatillo'] == 'venta':
    actions = actions.iloc[1:]

if actions.iloc[-1].loc['gatillo'] == 'compra':
    actions = actions.iloc[:-1]

actions
```

Date	Open	High	Low	Close	Volume	Balance	OBV	cruce	rsl	sigma	OBV_osc	gatillo
2011-05-31	5 938000	6 056000	5 910000	6 028000	16452500	16452500	118666000	0 068359	65 023007	0 028402	3 334472	compra
2012-01-18	5 338000	5 376000	5 250000	5 362000	6301000	6301000	80177500	-0.083017	44 723758	0 047041	0 003865	venta
2012-02-15	6 620000	6 882000	6 454000	6 720000	13809000	13809000	127319000	0 009988	65 937196	0 044627	2 782804	compra
2013-11-11	28 200001	29 084000	27 420000	28 940001	69988000	69988000	1228801000	-0.021696	37 997823	0 041509	0 003973	venta
2014-01-16	32 500000	34 540001	32 480000	34 194000	59797000	59797000	1481964000	0 028473	66 157930	0 044273	1 297222	compra
2014-04-22	41 271999	43 866001	41 001999	43 728001	49023500	49023500	1536108000	-0.017390	53 431157	0 037299	0 163592	venta
2014-06-16	41 352001	45 098000	41 251999	44 922001	66232000	66232000	1525634000	0 000109	66 662118	0 032350	-0 343382	compra
2014-10-15	44 000000	46 198002	43 464001	45 939999	45736500	45736500	1669980500	-0.021526	35 642476	0 027298	0 073664	venta
2015-04-24	44 099998	44 160000	43 602001	43 686001	12139000	-12139000	1398322000	0 003622	65 505542	0 021731	-0 452249	compra
2015-08-18	51 076000	52 189999	50 712002	52 144001	20975000	20975000	1484308500	-0.010631	53 378474	0 028123	0 730114	venta
2016-03-17	44 293999	45 700001	44 000000	45 276001	18914500	18914500	1474843000	0 002427	72 385048	0 035815	2 586702	compra
2016-05-20	43 397999	44 110001	43 270000	44 056000	45035500	45035500	1577211000	-0.016339	47 112499	0 022806	1 010787	venta
2016-12-23	41 599999	42 689999	41 542000	42 667999	23352500	23352500	1372508500	0 000129	69 928612	0 020594	0 089821	compra
2017-07-26	68 071999	69 099998	67 624001	68 769997	24104000	24104000	2097021500	-0.011907	53 076252	0 025141	1 352485	venta
2017-09-15	74 902000	76 000000	74 540001	75 961998	27102500	27102500	2205684500	0 018672	65 133157	0 022764	1 429404	compra
2017-10-24	67 760002	68 559998	67 232002	67 468002	22458500	22458500	2183333000	-0.011573	40 209417	0 019794	0 945891	venta
2018-01-22	69 879997	71 556002	69 839995	70 311995	31052000	31052000	2125269500	0 026197	65 322862	0 019258	0 216856	compra
2019-01-14	68 475998	68 500000	66 800003	66 879997	26236500	-26236500	1593132000	-0.013066	50 083332	0 036499	0 368072	venta
2019-07-12	47 950001	49 076000	47 942001	49 015999	46002500	46002500	699972000	0 009579	66 579924	0 029799	-0 782325	compra
2020-03-23	86 720001	88 400002	82 099998	86 858002	82272500	82272500	2388793000	-0.021433	34 312457	0 085488	0 584441	venta
2020-06-01	171 600005	179 800003	170 820007	179 619995	74697500	74697500	3244588000	0 209083	68 971247	0 046852	1 166281	compra
2020-11-16	408 929993	412 450012	404 089996	408 089996	26838600	-26838600	4464774200	-0.012067	46 553691	0 034105	0 834046	venta

Como ven, me queda la misma tabla que antes, pero ahora se me va la última fila que era la compra del 20-11-2020. Ojo hay algunos que dejan esa compra y simulan una venta forzada el último día de operaciones, pero está mal, digo mal, porque queremos backtestear la lógica completa con gatillo de compra y venta tal cual lo diseñamos y hacer eso supondría un último trade mentiroso porque no sale en las condiciones de gatillo, sino que sale forzado porque no tenemos mas datos en la serie.

Aclaración: No les dejo código aún para que copy-pasteen porque aún no llegamos a un nivel de profundidad o complejidad que lo amerite, por ahora son pocas líneas, no trabajamos con frameworks grandes como Django que si ameritaría que suba un repo y les deje link. Mi idea es en el tomo siguiente ya arrancar con este tipo de metodología así que no se desanimen que estas ya son las últimas páginas que les queda tippear linea a linea, pero créanme que es necesario, no es lo mismo como se fijan los contenidos de esta forma que copy-pasteando

Armado de Tabla de Trades

Una vez terminada la tabla de acciones, es decir de saber cuándo se activa cada gatillo, viene la parte de la tabla de trades, es decir trade por trade, desde la compra hasta la venta para construir el payoff más discreto es decir por trades

Hay muchas formas de hacer esto, vamos a valernos de las facilidades de pandas para construir una función bien pythónica, mi idea fue partir de la tabla de acciones y sacar por un lado las compras y por otro lado las ventas y armar una tabla única combinando una de cada, tipo fila1 con fila2, luego fila3 con fila4...y así sucesivamente.

Para ello tomo las filas pares por un lado y las impares por otro con el famoso slice de listas aplicado a las filas con `iloc[]` y luego concateno pares con impares, me quedaría algo así:

```
pares = actions.iloc[::2].loc[:,['Close']].reset_index()
impares = actions.iloc[1::2].loc[:,['Close']].reset_index()
trades = pd.concat([pares, impares], axis=1)

trades
```

	Date	Close	Date	Close
0	2011-05-31	6 028000	2012-01-18	5 362000
1	2012-02-15	6 720000	2013-11-11	28 940001
2	2014-01-16	34 194000	2014-04-22	43 728001
3	2014-06-16	44 922001	2014-10-15	45 939999
4	2015-04-24	43 686001	2015-08-18	52 144001
5	2016-03-17	45 276001	2016-05-20	44 056000
6	2016-12-23	42 667999	2017-07-26	68 769997
7	2017-09-15	75 961998	2017-10-24	67 468002
8	2018-01-22	70 311996	2019-01-14	66 879997
9	2019-07-12	49 015999	2020-03-23	86 859002
10	2020-06-01	179 619995	2020-11-16	408 089996

Obviamente que está incompleto le faltan varias cosas a saber:

- El nombre correcto de cada columna
- El rendimiento de cada trade
- La duración en días de cada trade
- Una clasificación de si el trade fue ganador o perdedor
- Un rendimiento acumulado del método

Podría requerir muchas cosas más, pero eso es lo básico, así que veamos el código para lograrlo de la forma más pythónica posible, como siempre.

Les decía el código para los ítems de la página anterior:

```
pares = actions.iloc[::2].loc[:,['Close']].reset_index()
impares = actions.iloc[1::2].loc[:,['Close']].reset_index()
trades = pd.concat([pares, impares], axis=1)

CT = 0
trades.columns = ['fecha_compra', 'px_compra', 'fecha_venta', 'px_venta']
trades['rendimiento'] = trades.px_venta / trades.px_compra - 1
trades['rendimiento'] -= CT
trades['dias'] = (trades.fecha_venta - trades.fecha_compra).dt.days

if len(trades):
    trades['resultado'] = np.where(trades['rendimiento'] > 0, 'Ganador', 'Perdedor')
    trades['rendimientoAcumulado'] = (trades['rendimiento']+1).cumprod()-1

trades
```

	fecha_compra	px_compra	fecha_venta	px_venta	rendimiento	dias	resultado	rendimientoAcumulado
0	2011-05-31	6.028000	2012-01-18	5.362000	-0.110484	232	Perdedor	-0.110484
1	2012-02-15	6.720000	2013-11-11	28.940001	3.306548	635	Ganador	2.830742
2	2014-01-16	34.194000	2014-04-22	43.728001	0.278821	96	Ganador	3.898832
3	2014-06-16	44.922001	2014-10-15	45.939999	0.022661	121	Ganador	4.009847
4	2015-04-24	43.686001	2015-08-18	52.144001	0.193609	116	Ganador	4.979798
5	2016-03-17	45.276001	2016-05-20	44.056000	-0.026946	64	Perdedor	4.818667
6	2016-12-23	42.667999	2017-07-26	68.769997	0.611746	215	Ganador	8.378216
7	2017-09-15	75.961998	2017-10-24	67.468002	-0.111819	39	Perdedor	7.329553
8	2018-01-22	70.311996	2019-01-14	66.879997	-0.048811	357	Perdedor	6.922979
9	2019-07-12	49.015999	2020-03-23	86.858002	0.772034	255	Ganador	13.039786
10	2020-06-01	179.619995	2020-11-16	408.089996	1.271963	168	Ganador	30.897876

Y ahí sí, como ven, ya tengo una tabla de trades bastante decente

Lo único que me empieza a hacer un poco de ruido es que ya metimos como 40 líneas de código y quedaron todas sueltas por ahí, y por esto está bueno que esta parte la vayan codeando par a par conmigo y no copy-pasteen, es una forma de darse cuenta que empezamos a necesitar algo para poner un orden a esto, en principio vamos a valernos de funciones, luego seguramente usemos clases y objetos, pero no quiero volverlos locos aun con nuevos conceptos, empecemos con funciones que son más sencillas

Simplemente vamos a “encapsular” en funciones el código que

- Sea repetitivo
- Que dependa de parámetros
- Que tenga un return
- Que en conjunto sea algo concreto fácil de separar del resto (ejemplo “bajar data”, “calcular indicadores”, “armar tabla de trades” etc)

Por ejemplo arranquemos por lo obvio, las funciones `getData()` para traer los datos de OHLC históricos y la función `getFeatures()` para calcular los indicadores que vayamos a usar en nuestro backtest:

```
def getData(ticker, data_from, data_to):
    data = yf.download(ticker, auto_adjust=True, progress=False,
                      start=data_from, end=data_to)
    return data

def getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60):
    data['Balance'] = np.where(data.Close>data.Close.shift(), data['Volume'],
                               -np.where(data.Close < data.Close.shift(), -data['Volume'], 0))

    data['OBV'] = data['Balance'].cumsum()
    dif = data['Close'].diff()
    win = pd.DataFrame(np.where(dif > 0, dif, 0), index=data.index)
    loss = pd.DataFrame(np.where(dif < 0, abs(dif), 0), index=data.index)
    ema_win = win.ewm(alpha=1/n_rsi).mean()
    ema_loss = loss.ewm(alpha=1/n_rsi).mean()
    rs = ema_win / ema_loss

    data['cruce'] = data.Close.rolling(fast).mean() / data.Close.rolling(slow).mean() - 1
    data['rsi'] = 100 - (100 / (1+rs))
    data['sigma'] = data.Close.pct_change().rolling(n_sigma).std()
    data['OBV_osc'] = (data.OBV - data.OBV.rolling(n_obv).mean()) / (data.OBV.rolling(n_obv).std())
    features = data.iloc[:, -4:].dropna()
    return features
```

Como ven es un copy-paste del código de las páginas anteriores, nada mas que metido adentro de funciones, una vez encapsulado todo esto en funciones, esto nos permite reutilizar todo el código de adentro metiendo las funciones dentro de otras funciones o dentro de bucles etc, corroboremos si funcionan nuestras funciones iniciales

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
```

Date	cruce	rsi	sigma	OBV_osc
2011-05-25	0.066759	60.657683	0.039441	2.154925
2011-05-26	0.066522	62.525084	0.039414	2.737019
2011-05-27	0.066726	62.790011	0.029056	2.922802
2011-05-31	0.068359	65.023007	0.028402	3.334472
2011-06-01	0.069083	55.266080	0.029319	2.884800
...
2020-12-23	0.247179	60.972635	0.041372	1.655826
2020-12-24	0.246281	63.025740	0.040430	1.692907
2020-12-28	0.245130	63.277422	0.040462	1.757391
2020-12-29	0.244585	63.595475	0.038955	1.795833
2020-12-30	0.244407	67.384055	0.039124	1.892423

2417 rows × 4 columns

Efectivamente, obtengo lo mismo que antes pero ahora todo mas prolijo con el código encapsulado dentro de funciones, vean que ahora solo voy a tipar esas 2 líneas de código cuando necesite esto, y si vuelan un poco ya se imaginarán que la linea de los features puede ir adentro de un bucle que

pruebe diferentes parámetros de cada cosa usamos, porque claramente no necesariamente usaremos el RSI con 15 velas podrían ser 20 o 10, en fin, lo bueno de ir encapsulando en funciones es que ya vamos aislando la parte funcional del código interno y de ahí en adelante se hace más fácil.

Bueno de la misma forma encapsulamos el resto:

Es importante encapsular "por funcionalidad" es decir que quede bien delegada en funciones cosas que son funcionalidades concretas, valga la redundancia, y separar lo más posible una funcionalidad de otra, es decir no mezclar el "traer data" con el "calcular features" que bien lo podríamos haber hecho, ¿por qué no?

Facil, imaginén que luego queremos recalcular todo el backtest con diferentes parámetros de indicadores, sería ridículo volver a bajar la data de internet cada vez que queremos cambiar un parámetro del RSI por decir algo, ya que la data seguirá siendo la misma, por eso conviene tener siempre separadas las funciones diferentes.

Entonces, función para traer acciones (compra/venta) y otra para armar tabla de trades:

```
def getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
              trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0):

    gatillos_compra = pd.DataFrame(index = features.index)
    gatillos_compra['cruce'] = np.where(features.cruce > trig_buy_cross, True, False)
    gatillos_compra['rsi'] = np.where(features.rsi > trig_buy_rsi, True, False)
    gatillos_compra['sigma'] = np.where(features.sigma > trig_buy_sigma, True, False)
    mascara_compra = gatillos_compra.all(axis=1)

    gatillos_venta = pd.DataFrame(index = features.index)
    gatillos_venta['cruce'] = np.where(features.cruce < trig_sell_cross, True, False)
    gatillos_venta['rsi'] = np.where(features.rsi < trig_sell_rsi, True, False)
    gatillos_venta['obv'] = np.where(features.OBV_osc > trig_sell_obv, True, False)
    mascara_venta = gatillos_venta.all(axis=1)

    data_aux = data.copy().dropna()
    data_aux['gatillo'] = np.where(mascara_compra, 'compra', np.where(mascara_venta, 'venta', ''))

    actions = data_aux.loc[data_aux.gatillo != ''].copy()

    actions['gatillo'] = np.where(actions.gatillo != actions.gatillo.shift(), actions.gatillo,'')
    actions = actions.loc[actions.gatillo != ''].copy()

    if actions.iloc[0].loc['gatillo'] == 'venta':
        actions = actions.iloc[1:]

    if actions.iloc[-1].loc['gatillo'] == 'compra':
        actions = actions.iloc[:-1]

    return actions
```

```
def getTrades(actions):
    pares = actions.iloc[::2].loc[:,['Close']].reset_index()
    impares = actions.iloc[1::2].loc[:,['Close']].reset_index()
    trades = pd.concat([pares, impares], axis=1)

    CT = 0
    trades.columns = ['fecha_compra','px_compra','fecha_venta','px_venta']
    trades['rendimiento'] = trades.px_venta / trades.px_compra - 1
    trades['rendimiento'] -= CT
    trades['dias'] = (trades.fecha_venta - trades.fecha_compra).dt.days

    if len(trades):
        trades['resultado'] = np.where(trades['rendimiento'] > 0 , 'Ganador' , 'Perdedor')
        trades['rendimientoAcumulado'] = (trades['rendimiento']+1).cumprod()-1

    return trades
```

Ahora fijense que con todo que fuimos viendo bien esto encapsulado tenemos todo lo visto en las últimas 10 páginas en algo así de tan solo 5 líneas (y mucho más fácil de digerir):

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                      trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)
trades = getTrades(actions)
trades
```

	fecha_compra	px_compra	fecha_venta	px_venta	rendimiento	dias	resultado	rendimientoAcumulado
0	2011-05-31	6 028000	2012-01-18	5 362000	-0.110484	232	Perdedor	-0.110484
1	2012-02-15	6 720000	2013-11-11	28 940001	3.306548	635	Ganador	2.030742
2	2014-01-16	34.194000	2014-04-22	43 728001	0.278821	96	Ganador	3.098632
3	2014-06-16	44.922001	2014-10-15	45.939999	0.022661	121	Ganador	4.009047
4	2015-04-24	43.686001	2015-08-18	52.144001	0.193609	116	Ganador	4.979799
5	2016-03-17	45 276001	2016-05-20	44 056000	-0.026946	64	Perdedor	4.818667
6	2016-12-23	42 667999	2017-07-26	68.769997	0.611746	215	Ganador	8.378216
7	2017-09-15	75.961998	2017-10-24	67 468002	-0.111819	39	Perdedor	7.329653
8	2018-01-22	70.311996	2019-01-14	66 879997	-0.048811	357	Perdedor	6.922979
9	2019-07-12	49 015999	2020-03-23	86 858002	0.772034	255	Ganador	13.039786
10	2020-06-01	179 619995	2020-11-16	408 089996	1.271963	168	Ganador	30.897076

Vean que ahora toda la parametrización quedó ahí a la vista y solo invocando a las funciones ya obtengo todo procesado y listo.

Tabla de Resultados Básicos

Bueno, siguiente paso empezar a reportear resultados

Lo mas básico sería ver la cantidad de trades ganadores y perdedores:

```
trades.groupby('resultado').size()

resultado
Ganador      7
Perdedor     4
dtype: int64
```

Luego los promedios de los ganadores y de los perdedores:

	px_compra	px_venta	rendimiento	dias	rendimientoAcumulado
resultado					
Ganador	57 260856	104.924285	0.922483	229.428571	9.719300
Perdedor	49.394499	45.941500	-0.074515	173.000000	4.740179

Podríamos ver rápidamente también algo referido a la duración, en este caso uso la función de pandas describe() así veo todos los cuartiles mas allá de la media y de paso en la misma tabla veo el conteo ganadores/perdedores:

```
trades.groupby('resultado').describe().dias
```

	count	mean	std	min	25%	50%	75%	max
resultado								
Ganador	7.0	229.428571	187.771898	96.0	118.50	168.0	235.00	635.0
Perdedor	4.0	173.000000	149.637339	39.0	57.75	148.0	263.25	357.0

Como ven empieza a tomar forma, parece todo muy bonito

- Mas ganadores que perdedores (7 a 4)
- Las ganadoras, en promedio ganan más (92.2%) que lo que pierden las perdedoras (-7.4%)

Pero, no se si dan cuenta, tengo muchas preguntas que, al menos a mí, me generan dudas

- ¿Es mejor esto que el buy&Hold? (A lo mejor el método da bueno porque es LONG en TSLA q en ese período subió un montón)
- ¿Vale la pena ese retorno por el riesgo asumido en el método?
- ¿Cuántos días estuvo invertido el método? ¿Qué podría haber hecho con el capital el tiempo desinvertido?
- ¿Si lo pruebo con mas trades se mantiene ese sesgo es fue suerte que esos pocos trades dio muy bien?
- ¿Si lo pruebo con otros activos?
- ¿Si le cambio parámetros?

Bueno, todo eso y mucho mas vamos a empezar a respondernos en las siguientes etapas, pero al menos en esta etapa la idea era poder tener un primer dashboard de resultados del método de trading backtesteado con todos los datos pasados disponibles.

Y falta también un resultado que hasta ahora no mencionamos ni calculamos y en finanzas resulta que es bastante importante: "el ROI" o la TIR o la TEA, o el CAGR, o como lo llamen, me refiero a anualizar los retornos obtenidos, y dado que es un método que entra y sale, hay que contabilizar el tiempo neto invertido y al resultado acumulado hay que anualizarlo, en fin esta vez lo voy a meter todo en una función de una.

```

def resumen(trades):
    if len(trades):
        resultado = float(trades.iloc[-1].rendimientoAcumulado-1)
        agg_cant = trades.groupby('resultado').size()
        agg_rend = trades.groupby('resultado').mean()['rendimiento']
        agg_tiempos = trades.groupby('resultado').sum()['dias']
        agg_tiempos_medio = trades.groupby('resultado').mean()['dias']

        r = pd.concat([agg_cant,agg_rend, agg_tiempos, agg_tiempos_medio], axis=1)
        r.columns = ['Cantidad', 'Rendimiento x Trade', 'Días Total', 'Días x Trade']
        resumen = r.T

        try:
            t_win = r['Días Total']['Ganador']
        except:
            t_win = 0

        try:
            t_loss = r['Días Total']['Perdedor']
        except:
            t_loss = 0

        t = t_win + t_loss
        tea = (resultado +1)**(365/t)-1 if t > 0 else 0

        metricas = {'rendimiento':round(resultado,4), 'dias_in':round(t,4), 'TEA':round(tea,4)}
    else:
        resumen = pd.DataFrame()
        metricas = {'rendimiento':0, 'dias_in':0, 'TEA':0}

    return resumen, metricas

```

Lo de los try/except es por si no hay ningún trade ganador o perdedor o por si directamente no ejecuta ningún trade el backtest (tiene que estar preparada para todo la función)

La TEA o tasa efectiva anualizada la calculo con el rendimiento total acumulado y el "n" o cantidad de años transcurridos, para lo cual necesito la cantidad de días totales "invertidos" ya que no cuenta cuando no estamos invertidos porque en ese momento el capital estaría libre para otras inversiones.

Bueno, finalmente aplicando este mini dashboard obtengo lo siguiente:

```

data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                      trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

trades = getTrades(actions)
resumen(trades)

(resultado      Ganador      Perdedor
 Cantidad       7.000000   4.000000
 Rendimiento x Trade  0.922483 -0.074515
 Días Total     1606.000000 692.000000
 Días x Trade   229.428571 173.000000,
 {'rendimiento': 29.8979, 'dias_in': 2298, 'TEA': 0.7245})

```

Como ven la salida es un DataFrame por un lado y el diccionario con los datos de rendimiento acumulado, días invertido y TEA.

Cabe aclarar que todos los valores de tasas están expresados en formato decimal, es decir esto tiene una TEA del 72.45%, parece enorme pero caben preguntarse muchas cosas aún, entre ellas:

- ¿Es mejor esto que el buy&Hold?

Esto se responde comparando con el buy&Hold en el mismo período.

- ¿Vale la pena ese retorno por el riesgo asumido en el método?

Para esto, tengo que usar ratios de riesgo que ya veremos en la siguiente etapa

- ¿Si le cambio parámetros?

Esto se responde en la etapa de parametrización

- ¿Si lo pruebo con otros activos?

Esto es parte de la etapa de contrastes, validaciones y portabilidad

Como ven esto recién empieza pero ya podemos ir viendo algunas cosas, por ejemplo

A favor:

- El método es rentable
- Tiene mas trades ganadores que perdedores
- La asimetría del rendimiento de los ganadores sobre el de los perdedores es muy favorable

En contra:

- Son apenas 11 trades (es un "n" muy chico)
- Aun no tenemos ni idea si en el mismo período a TSLA le fue mejor o peor
- Aun no tenemos idea de si el riesgo asumido valió la pena contra el retorno obtenido
- No tenemos idea si funciona con otros activos

Por ejemplo ya que tenemos todo empaquetado en funciones, probemos por probar algún otro activo:

```
data = getData(ticker='AMZN', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                     trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

trades = getTrades(actions)
resumen(trades)

(resultado          Ganador      Perdedor
 Cantidad           7.000000    5.000000
 Rendimiento x Trade  0.475416   -0.086165
 Dias Total         2390.000000  227.000000
 Dias x Trade       341.428571   45.400000,
 {'rendimiento': 5.3837, 'dias_in': 2617, 'TEA': 0.295})
```

Me sigue dando positivo con AMZN que fue lo único que cambié, pero no tan bueno como con TSLA

```

data = getData(ticker='AAPL', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                     trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_sigma=0.01,
                     trig_sell_low=0)

trades = getTrades(actions)
resumen(trades)

(resultado      Ganador
Cantidad       8.000000
Rendimiento x Trade 0.385457
Dias Total    2771.000000
Dias x Trade   346.375000,
{'rendimiento': 8.1392, 'dias_in': 2771, 'TEA': 0.3384})

```

Como ven 8 a 0, espectacular, pero ojo, son solo 8 trades

Esto me lleva a decir, "ok" probemos con los 500 del SP500 y a promedio 10 trades por cada uno tendremos unos 5000 trades, mas que suficiente, pero es medio una trampa porque probablemente, al ser una estrategia de momentum, todos repitan trades en el mismo momento ya que los activos guardan cierta correlación, pero bueno, al menos sería mejor que andar sacando conclusiones con 8 o con 11 trades "sueltos". Igual como les decía antes ya es parte de otro análisis de una etapa posterior.

Para ir finalizando esta etapa de "idea rápida" de resultados, podríamos nombrar algunas métricas mas que podrían utilizarse en este reporte rápido, insisto, es solo un primer vistazo, falta mucho camino por recorrer aún pero es solo el primer "dashboard" de resultados, yo diría que lo mas importante es esa TEA o tasa de retorno anualizada porque tiene en cuenta el resultado acumulado mas el tiempo invertido, pero también podría agregarse en esta etapa previa lo siguiente:

- % de trades positivos y negativos y Esperanza matemática: Esto nos va a permitir saber si el método es mas ganador que perdedor, y nos dará un vistazo rápido de la asimetría de los trades ganadores vs perdedores. Tiene mas sentido en estrategias de contrarian donde pesan mas los stopLoss y los trades son mas cortos y mas simétricos para ambos lados, en cambio en las de momentum como esta
- Costos transaccionales: En este caso al ser trades largos con medias de rendimiento por trade de varios puntos porcentuales, la verdad, no inciden, pero en temporalidades mas cortas donde hay mucha mas cantidad de operaciones y los rendimientos medios son menores al 1%, estos datos de costos transaccionales empiezan a jugar un roll más importante, en tal caso el cálculo de "Esperanza matemática" o rendimiento medio de todos los trades (positivos y negativos" empieza a ser mas necesario porque debe ser mayor al costo transaccional y muchas veces el método puede parecer rentable pero teniendo en cuenta estos costos, deja de serlo
- Costo de oportunidad: Básicamente la tasa de interés (de mínima), es decir, no debemos asumir que si no se está invertido es lo mismo que 0%. Porque el dinero en ese tiempo tiene un costo. Ahora hay toda una discusión de si tomar como costo de capital "la tasa libre de riesgo" soy un tanto crítico de este enfoque porque el dinero que destinamos al trading es "dinero de riesgo" y debemos asumir que el costo de oportunidad es mucho mas alto que la "tasa libre de riesgo" que sería un concepto que aplica mas a la comparación contra inversiones en bonos o ese tipo de asset, pero no en trading algorítmico donde ya asumimos otro nivel de riesgo, yo suelo tomar el CAGR medio del SP500 como referencia de costo de capital pero insisto es un tema polémico cada cual lo verá como quiera, solo menciono el tema.

Etapa de Análisis y Métricas

Bueno, viene la parte central de todo backtest que es donde aparecen

- Ratios de performance
- Métricas de referencia
- Ratios de Riesgo

Comencemos por usar una librería, pero vamos a ir calculando a mano luego cada cosa y sobre todo a ir entendiendo lo conceptual de cada métrica, lo primero que necesitamos es pasar de una tabla de trades a un payoff diario del método

De la tabla de trades al payoff diario

Acá viene la parte complicada, hasta ahora teníamos un enfoque matricial del asunto, es decir podíamos filtrar "saltos discretos de tiempo" entre una entrada y una salida, es decir comprar a "X" y luego de tal gatillo, vender a "Y", luego podíamos armar una tabla de trades con todas las entradas y salidas y resultados y calcular un resultado final.. Fenómeno, pero es incompleto ¿por qué? Porque no contempla lo que pasa en el medio

No es lo mismo un trade de comprar \$5 y vender a \$6 si tuvo diez velas intermedias de \$5.1, \$5.2, \$5.3... y así hasta \$5.9 y \$6... Que haber pasado de \$5 a \$6 con 10 velas entre las cuales bajó a \$3 y luego pasó por \$8 para terminar en \$6

¿Qué pasa? Que el camino importa, no solo por todo lo que podamos agregar de gatillos instantáneos como stopLoss, takeProfits en spikes etc, sino porque la volatilidad importa, los drawdowns importan, etc, no es solo cuestión de mirar el resultado final porque en el día a día hay que sobrellevar la volatilidad

Esquema discreto de tabla de trades

Trades	Fecha Compra	Precio Compra	Fecha Venta	Precio Venta	P&L	Días
Trade 1	fecha 1	X	fecha 2	Y		
Trade 2						
Trade 3						

Esquema continuo basado en eventos

Esquema Comprando y Vendiendo al cierre

Fecha	Signal	Estado	Precio Cartera	Variación diaria
	--	Out		--
fecha 1	BUY	Out		--
	--	In		xx %
	--	In		xx %
	--	In		xx %
fecha 2	SELL	In		xx %
	--	Out		--

Enfoque Event Driven

Ahora que ya sabemos a dónde queremos llegar, tenemos que armar una iteración que vaya viendo día a día como acomoda los datos de la tabla de trades para pasar a un esquema conducido por eventos (señales) que me determinan, día a día, o mejor dicho vela a vela, o bien podría ser tick by tick, el P&L de cada instante (de hecho debería ser tick by tick pero veremos esto al final del libro).

La forma de armarlo es más engorrosa, debemos hacer una iteración que contemple el "tiempo continuo" y lo pongo entre comillas porque es medio una trampa, las velas no son un "tiempo continuo" por más que sean de 1 minuto o de segundos, sino que son intervalos de tiempo discreto, pero consideremos cada vela como "un instante" (hay que hacer esta abstracción para entenderlo mejor)

Primero vamos a poner un ejemplo trivial para comprender el esquema iterativo porque es bastante más complejo que lo que veníamos haciendo con pandas desde un punto de vista matricial.

Como ven en el ejemplo, tengo un "set de datos" teórico de 10 velas con sus 10 "pct_changes" diarios. Les sugiero jugar con el ejemplo trivial cambiando datos para chequear el funcionamiento, ojo, es solo mi forma de resolver la iteración, seguramente hay formas mejores.

Esquema asumiendo compra al cierre:

```
signals = ['-', 'compra', '-', '-', 'venta', '-', 'compra', '-']
pct_changes = [1,2,3,4,5,6,7,8]
i, results = 1, [0]

while i < len(signals):
    if signals[i-1] == 'compra':
        j = i
        while j < len(signals):
            results.append(pct_changes[j])
            j +=1

            if signals[j-1]== 'venta':
                i = j; break
            if j == len(signals):
                print('Ojo que queda compra abierta..')
                i = j; break
    else:
        results.append(0)
        i +=1

results, len(results)

Ojo que queda compra abierta..
([0, 0, 3, 4, 5, 0, 0, 8], 8)
```

Como sea que lo piensen siempre caen en los bucles anidados ¿Por qué? Simple, porque hay que ir recorriendo vela por vela (primer bucle), y una vez encontrada la señal hay que entrar en un nuevo bucle que a partir de ahí recorra vela por vela adentro buscando la salida.

Se que esto es medio "mind blowing" pero toméngase su tiempo,

Les dejo el link a esta herramienta online que está buenísima para debuggear el paso a paso:

- <https://pythontutor.com/>

En realidad, cualquier IDE tiene herramientas de debug, pero son menos intuitivas para usar y depende del IDE de cada uno así que no daba poner cada ejemplo acá, pero esto es más universal

El Paso a Paso

Siguiendo con nuestro ejemplo trivial de 8 velas para entender conceptualmente los bucles anidados y las iteraciones, vamos a usar la herramienta de debug que les digo online, me gusta porque es muy intuitiva, una vez que se familiaricen con hacer un debug, obviamente van a usar las herramientas integradas de los IDEs para esto, pero como primer debug me parece mejor esto.

Yo lo fui ejecutando online el ejemplo y en el exacto momento que detecta la primera señal de compra estaba este estado, vean que la herramienta me muestra todo, el estado de cada variable al momento de cada paso, y va mostrándome a la izquierda de todo con esas "flechitas" la línea de código por la que está evaluando. Es un "paso a paso" digamos



Python 3.6
(Known limitations)

```
signals = ['-', 'compra', '-', '-', 'venta', '-', 'compra', '-']
pct_changes = [1,2,3,4,5,6,7,8]
i, results = 1, [0]

while i < len(signals):
    if signals[i-1] == 'compra':
        j = i
        while j < len(signals):
            results.append(pct_changes[j])
            j += 1

            if signals[j-1] == 'venta':
                i = j; break
            if j == len(signals):
                print('ojo que queda compra abierta..')
                i = j; break
    else:
        results.append(0)
    i += 1

results = len(results)
```

Print output (drag lower right corner to resize):

	list	1	2	3	4	5	6	7	8
signals	["-",	"compra"	"-",	"-",	"venta"	"-",	"compra"	"-",
pct_changes	[1	2	3	4	5	6	7	8
i	1	2	3	4	5	6	7	8	
j	2	3	4	5	6	7	8		
results	[0	1	2	3	4	5	6	7
		0	0	3					

Frames Objects

Global frame

Line that just executed

next line to execute

<< First < Prev Next >> Last >>

Step 13 of 45

Customize visualization

Bien, si llegaron acá y no comprendieron bien el ejemplo trivial, no pasa nada, puede seguir adelante porque no es algo q vayamos a necesitar, lo ideal sería no solo que lo entiendan, sino que puedan cranear sus propias soluciones y mejorar la que yo hice, hacerla mas pythónica etc, yo la hice con el chip de otros lenguajes que tienen sus diferencias con Python así que de seguro encontrarán una forma mejor si se lo proponen.

Bueno, dada por superada la idea que entendieron la lógica de la función con el ejemplo trivial, aplíquemolas al ejemplo real:

La función devuelve un df igual al pasado como argumento, pero agregando la columna <strategy>. Dicha columna tiene un 0 o el valor de pct_change, dependiendo si está comprado o no. Para saber si un día se está comprado o no, se toma la señal del día anterior (precios de cierre)

```
def eventDrivenLong(df):
    df['pct_change'] = df['Close'].pct_change()
    signals = df['gatillo'].tolist()
    pct_changes = df['pct_change'].tolist()

    total = len(signals)
    i, results = 1, [0]

    while i < total:
        if signals[i-1] == 'compra' :
            j = i
            while j < total:
                results.append(pct_changes[j])
                j +=1
                if signals[j-1]== 'venta' :
                    i = j
                    break
                if j == total:
                    i = j
                    print('Ojo que queda compra abierta..')
                    break
        else:
            results.append(0)
            i +=1

    result = pd.concat ([df,pd.Series(data=results, index=df.index)], axis=1)
    result.columns.values[-1] = "strategy"
    return result
```

Como ven es exactamente igual a la función trivial que hice de ejemplo, solo que con la serie de datos reales

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                     trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

payoff = eventDrivenLong(data)
results = payoff.iloc[:, -2: ].add(1).cumprod()
results

Ojo que queda compra abierta..
```

En la página que viene pego el resultado porque acá no me entra pero aprovecho el espacio para reflexionar algo interesante.. Vean que yo le paso "data" a mi función eventDrivenLong() pero ese "data" no fue pasado como argumento a las funciones que calculaban los features y las actions, ni fue devuelto como return de esas funciones intermedias, sin embargo adentro de las mismas esa matriz fue modificada agregándose los features y gatillos, esto es posible (en Python) porque los dataframes son objetos mutables, no importa el tecnicismo, el hecho es que "**cuando un objeto mutable es modificado dentro de una función, queda modificado afuera**"

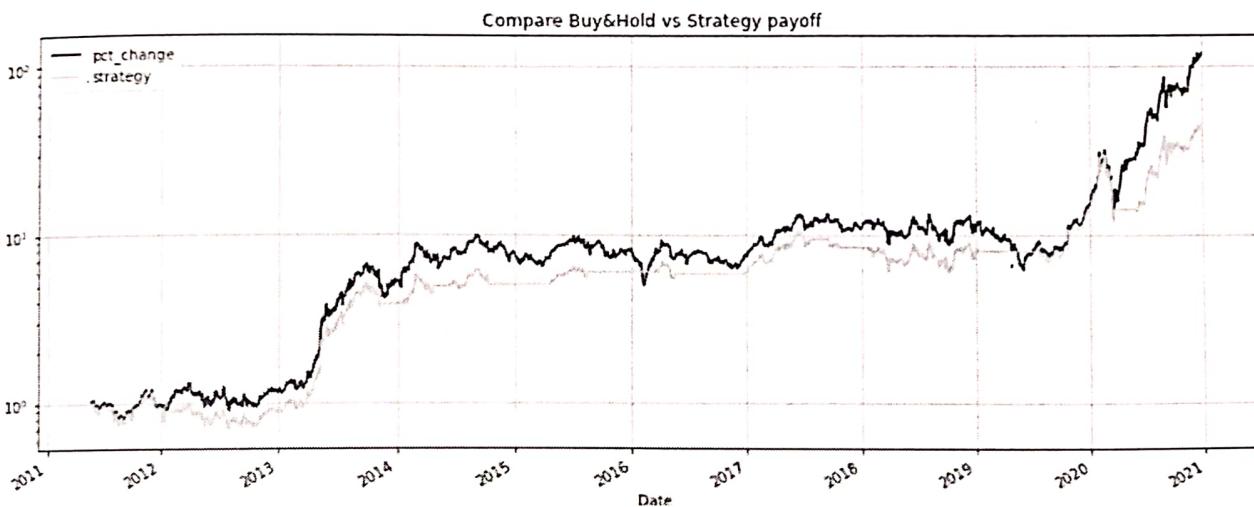
Ahora si les pego el resultado de mi eventDrivenLong().

Date	pct_change	strategy
2011-05-25	NaN	1 000000
2011-05-26	1 017253	1 000000
2011-05-27	1 019669	1 000000
2011-05-31	1 040028	1 000000
2011-06-01	0 984127	0 946251
...
2020-12-23	111.452723	42 085313
2020-12-24	114 177022	43 114027
2020-12-28	114 508282	43 239113
2020-12-29	114 905105	43 388956
2020-12-30	119 872331	45 264615

2417 rows x 2 columns

Y ya que estamos vamos a graficarlo a ver como se ve, recordemos que a priori los primeros resultados arrojaban métricas muy prometedoras, pero miren lo que pasa..

```
t = 'Compare Buy&Hold vs Strategy payoff'  
results.plot(figsize=(15,6), title=t, grid=True, logy=True)
```



Si observamos bien, casi siempre la estrategia performa peor que el buy & Hold, solo recupera en algunas caídas del activo, pero rápidamente vuelve a ser superada por el payoff del buy & hold, con lo cual, ya tenemos un primer indicio de que no siempre buenas métricas de performance significan que sea una buena idea de trading.

La explicación es bastante obvia, y es que TSLA en ese período fue super alcista, con lo cual, cualquier estrategia de comprar y vender, por mas que pongamos a un mono a apretar el botón, siempre va a dar positiva.

Bueno, cerrando por ahora el capítulo del modelo evento-driven de armar un payoff, voy a remarcar que es en este momento en donde van los gatillos de stopLoss y TakeProfit, quiero decir dentro de esa función. Se que es un tema mas engorroso que lo que veníamos haciendo todo prolíjito con pandas, pero es cuestión de acostumbrarse a los punteros recorriendo una matriz, tengan paciencia y recuerden el paso a paso que a la larga se hace mas facil.

Métricas de Riesgo

Bueno, viene una parte larga pero super interesante, como venimos viendo, no alcanza solo con saber que es rentable, sino comparado contra algo, pero tampoco alcanza con eso, ya dijimos que no es lo mismo sacar un x% de ganancia con una volatilidad tranquila que con una montaña rusa de subidas y bajadas, de alguna forma tenemos que medir o cuantificar el riesgo y ahí viene la discusión del siglo

Digo así porque los libros clásicos cuantifican el riesgo por la volatilidad y muchos (me incluyo) consideramos que la volatilidad no es una buena medida del riesgo

Ahora pensemos un poco, no es que la volatilidad no tenga nada que ver, pero es demasiado incompleta
Imaginen estar comprado en estos dos bichos:



Son simulaciones de GBM, que, como vemos, el de la derecha tiene mayor sigma, es decir, mayor volatilidad, sin embargo, el de la izquierda parece más arriesgado. ¿Qué es lo que da esa sensación de riesgo? Para mí se resume en "LOS DRAWDOWNS" pero podrían usarse otros ratios, como el ratio mu/sigma o cosas similares, o como veremos ratios de colas, o vAr etc etc, eso es lo que vamos a ver un poco más en profundidad ahora

Vamos a intentar encontrar los mejores ratios para medir el riesgo y la comparativa para decidir si el método de trading tiene sentido o no, pero no solo mirando si es rentable sino teniendo una mirada más amplia.

Para ello vamos a utilizar la librería quantstats que usamos en el t[6] pero vamos igualmente a calcular todo o casi todo a mano para ver y repasar mejor algunos conceptos que no me gustan del todo como están calculados en la librería así que lo vamos a complementar, pero usamos la librería de referencia ya que es una muy buena guía con muchas métricas muy usadas todas ahí a un clic.

Por si no recuerdan tienen que instalar la librería con:

```
! pip install QuantStats
```

Y la importamos junto al módulo extend_pandas() para poder usar desde pandas los métodos de la librería

```
import quantstats as qs
qs.extend_pandas()
```

Listo, ahora a la acción, necesitamos el payoff de nuestra estrategia y el payoff de un benchmark, en principio tomemos de benchmark el mismo activo Buy&Hold. Tomo exactamente el mismo modelo que veníamos trabajando, le cambio levemente el parámetro de cruce de medias para mostrar algo específico luego.

```

data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=10, slow = 40)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                      trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

payoff = eventDrivenLong(data)
results = payoff.iloc[:, -2: ].add(1).cumprod()

qs.reports.full(results['strategy'], benchmark=results['pct_change'])

```

Performance Metrics

	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
Volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.83	0.39
Kurtosis	14.29	6.13
Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.99%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (cVaR)	-4.02%	-5.49%
Payoff Ratio	1.17	1.17
Profit Factor	1.4	1.24
Common Sense Ratio	1.65	1.44
CPC Index	0.89	0.76
Tail Ratio	1.18	1.15
Outlier Win Ratio	10.74	4.1
Outlier Loss Ratio	3.62	3.67
MTD	22.41%	22.41%
3M	35.76%	64.95%
6M	183.26%	244.17%
YTD	474.97%	730.42%
1Y	480.0%	737.69%
3Y (ann.)	116.49%	123.78%
5Y (ann.)	63.51%	70.96%
10Y (ann.)	55.67%	64.28%
All-time (ann.)	55.67%	64.28%
Best Day	24.4%	24.4%
Worst Day	-21.06%	-21.06%
Best Month	81.07%	81.07%
Worst Month	-23.75%	-22.43%
Best Year	474.97%	730.42%
Worst Year	-12.63%	-10.97%
Avg. Drawdown	-7.47%	-9.03%
Avg. Drawdown Days	51	46
Recovery Factor	181.73	192.72
Ulcer Index	inf	1.01
Avg. Up Month	16.45%	20.48%
Avg. Down Month	-8.8%	-8.81%
Win Days %	54.39%	51.85%
Win Month %	64.0%	55.17%
Win Quarter %	63.64%	53.85%
Win Year %	80.0%	80.0%
Beta	0.54	-
Alpha	0.17	-
None		

Como ven, por si no recuerdan esta librería del t[6] nos devuelve una serie larga de indicadores, lo cuales veremos a continuación uno por uno, pero básicamente nos devuelve 2 tipos de indicadores:

- Indicadores de Riesgo
- Indicadores de Performance

Y todo esto siempre comparando a la estrategia con el benchmark (en nuestro caso elegimos que el benchmark sea el Buy&Hold)

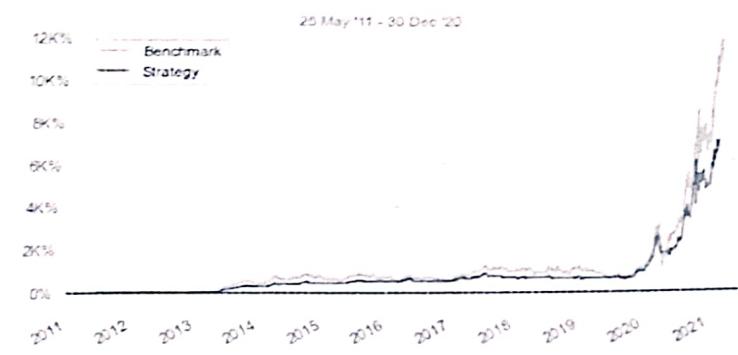
5 Worst Drawdowns

	Start	Valley	End	Days	Max Drawdown	99% Max Drawdown
1	2011-11-17	2012-08-02	2013-04-22	522	-38.113551	-35.148640
2	2017-06-26	2019-08-23	2019-11-03	655	-37.029987	-35.228853
3	2020-02-20	2020-04-21	2020-07-01	132	-37.027594	-35.688763
4	2020-09-01	2020-09-08	2020-12-07	97	-33.735353	-26.497031
5	2016-04-07	2016-12-21	2017-02-09	308	-21.697938	-20.303570

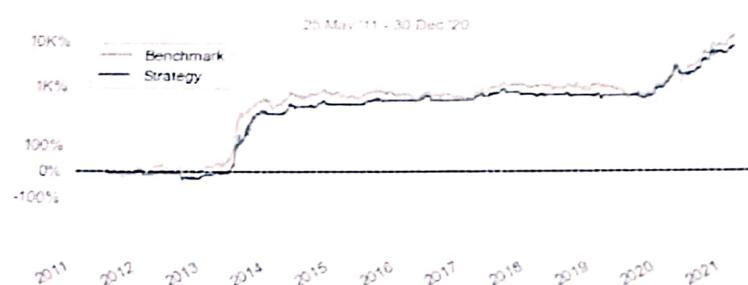
Nos ofrece un muy interesante cuadro con los 5 peores drawdowns del periodo para nuestra estrategia

Strategy Visualization

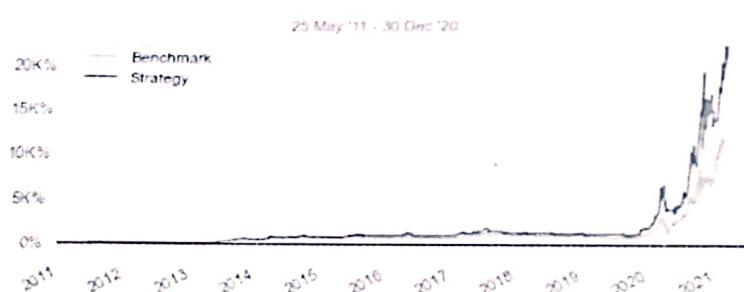
Cumulative Returns vs Benchmark



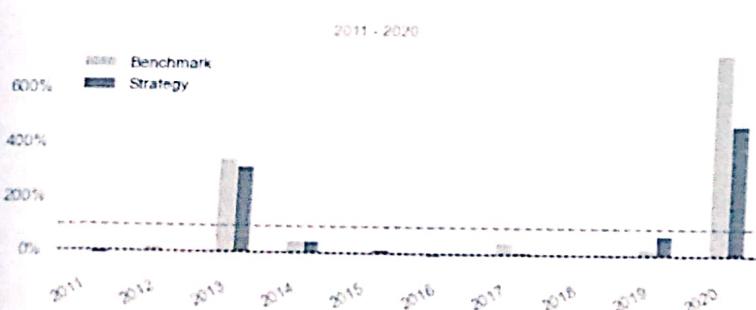
Cumulative Returns vs Benchmark (Log Scaled)



Cumulative Returns vs Benchmark (Volatility Matched)



EOY Returns vs Benchmark



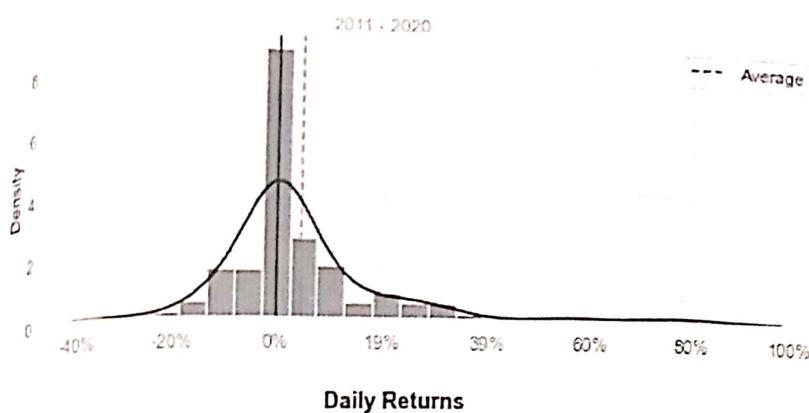
Obviamente no puede faltar la gráfica comparativa del payoff de la estrategia comparada con el benchmark, que fue lo primero que hicimos nosotros antes

Misma gráfica, pero en escala logarítmica, ahora lo interesante es la que viene a continuación.

Esta gráfica que sigue ajusta el retorno por volatilidad, es decir que castiga aquel retorno que haya tenido mas volatilidad, equiparando de alguna forma por riesgo, vean que de este modo "se da vuelta" es decir parece mas rentable la estrategia que el buy & Hold, esto es porque si bien dan retornos muy parejos, cuando se miden "por unidad de volatilidad" el retorno de la estrategia es superior.

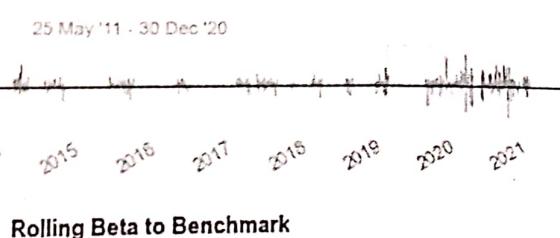
Una comparación año a año también es muy interesante siempre para ver aquellos años clave (por ejemplo, años de grandes caídas del mercado o similares)

Distribution of Monthly Returns



Gráfica de distribución de retornos mensuales, siempre es interesante por supuesto que la media esté a la derecha del cero, pero también ver si se debe a uno pocos meses muy fuera de escala o si la distribución es pareja como en este caso

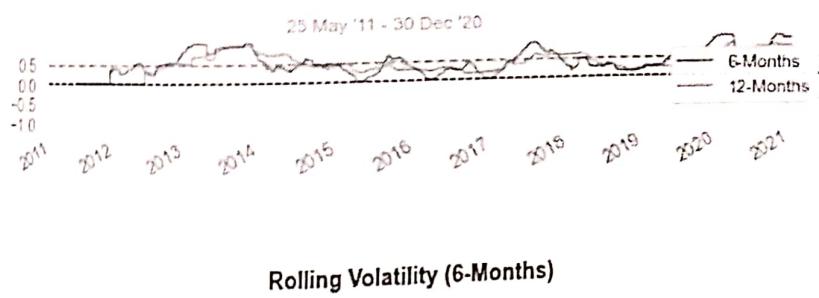
Retornos diarios, en este caso es mas estocástica la forma y muy difícil de interpretar en mi opinión.



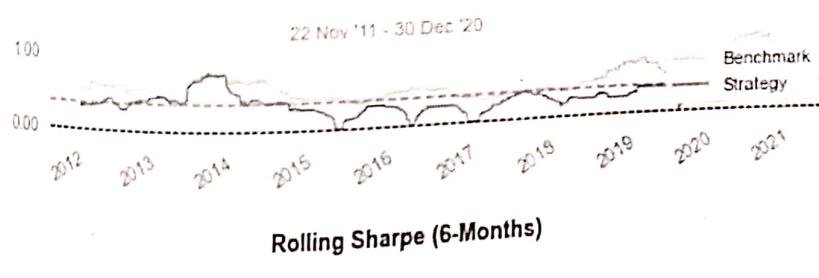
Rolling Beta to Benchmark

El Beta viene a representar el exceso de volatilidad respecto al benchmark, beta =1, significa que la estrategia tiene la misma volatilidad que el buy&Hold, menor a 1 significa que por cada 1% que se mueva el Buy&Hold se moverá menos la estrategia

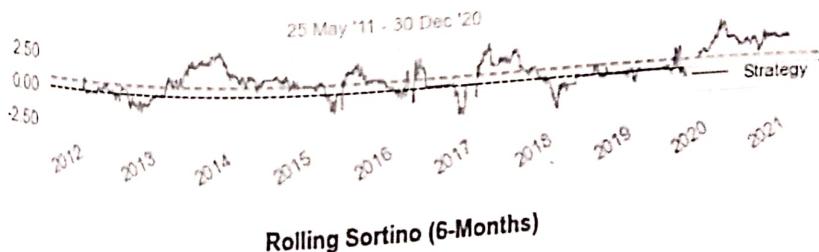
La volatilidad son los desvíos estándar de los retornos diarios.



Rolling Volatility (6-Months)



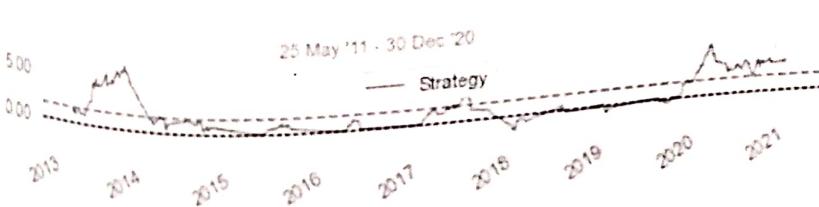
Rolling Sharpe (6-Months)



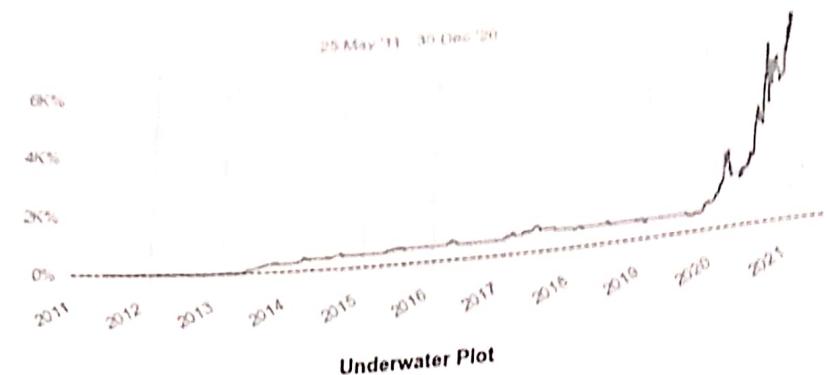
Rolling Sortino (6-Months)

El sharpe Ratio (a tasa libre de riesgo=0%) es el cociente entre rendimiento y volatilidad.

El Sortino Ratio es similar al sharpe, pero solo consideramos la volatilidad de los días que baja



Top 5 Drawdown Periods



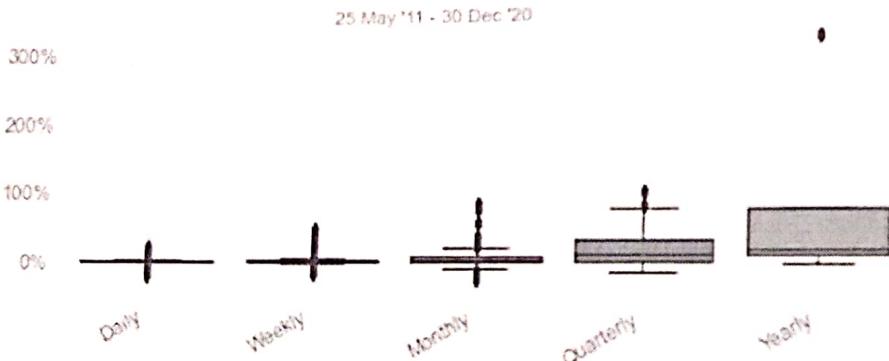
Aquí vemos los 5 peores drawdowns, pero en forma visual, antes lo hablamos visto en una tabla, a mi gusto esto tendría que estar en escala logarítmica, pero al menos se puede observar bien la duración de los drawdowns.

Este gráfico muestra la profundidad de los drawdowns, la va a apreciar mejor cuando lo comparemos con el Buy&Hold como la bondad de la estrategia para recortar estos drawdowns.

Monthly Returns (%)

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
2011	0.00	0.00	0.00	0.00	0.00	-6.66	0.00	0.00	-1.67	11.47	-12.74	0.15
2012	0.00	5.06	11.46	-10.56	0.00	0.00	-23.75	4.01	2.66	-3.93	20.23	0.00
2013	10.75	-7.14	7.75	22.99	81.07	9.82	25.07	25.86	14.42	-14.95	0.00	0.00
2014	6.11	34.95	-14.85	8.13	0.00	6.88	6.98	20.78	8.44	0.00	0.00	0.00
2015	0.00	0.00	0.00	3.01	10.05	6.96	-0.79	4.19	0.00	0.00	0.00	2.35
2016	0.00	0.00	10.09	4.78	-13.22	0.00	0.00	0.00	0.00	0.00	0.00	-9.35
2017	17.60	-0.77	-0.05	12.85	8.58	6.04	-9.51	0.00	-10.19	2.06	0.00	-5.04
2018	13.80	-5.31	0.00	0.00	0.00	3.27	-5.91	0.00	0.00	1.80	0.00	-26.79
2019	17.2	0.00	0.00	0.00	0.00	0.00	-1.42	-6.62	6.76	30.74	4.77	22.41
2020	55.42	2.68	5.05	37.1	6.79	29.32	32.50	74.15	-13.91	-0.55	20.38	22.41

Return Quantiles



No podría faltar un heatmap estacionalidad mensual, es un tipo de gráfico que de un simple vistazo dice mucho y no hace ni falta explicar.

Y finalizamos con un BoxPlot. Lo mismo que en el gráfico de drawdowns la gracia de este boxplot es compararlo con el del benchmark, ya lo veremos más adelante.

Ahora empecemos a desarmar un poco el análisis ratio por ratio a mano.

Rendimientos Lineales y Logarítmicos

Antes de empezar a calcular todo a mano, conviene tener tres series, tanto para la estrategia, como para el benchmark, ellas son:

- Los payoffs: Una base de precios base 100 o base 1 inicial
- Los rendimientos diarios lineales
- Los rendimientos diarios logarítmicos

Van a haber cálculos para los que nos va a ser más cómodo usar unos u otros, por lo que conviene calcular todo de entrada así los tenemos luego a mano

```
strategy = results['strategy']
benchmark = results['pct_change']

strategy_log = np.log(strategy / strategy.shift())
benchmark_log = np.log(benchmark / benchmark.shift())

strategy_lin = strategy.pct_change()
benchmark_lin = benchmark.pct_change()
```

Ya con estas 6 series a mano se hace todo el resto más fácil.

CAGR

Por si algún lector no lo sabe CAGR es la sigla de "Tasa de crecimiento anual compuesta" (en inglés obvio), básicamente se calcula con la famosa fórmula de crecimiento compuesto, donde " V_f " es el valor final de la cartera, " V_i " el valor inicial, " r " la tasa de crecimiento y " t " la cantidad de períodos (en la misma unidad que se mide la tasa, o sea en este caso, años)

$$\frac{V_f}{V_i} = (1 + r)^t$$

Calculamos entonces el CAGR tanto para la estrategia nuestra como para el benchmark del buy&Hold

```
(strategy_lin.index[-1]-strategy_lin.index[0]).days / 365
9.608219178082193

retorno_acum = (strategy_lin+1).prod()-1
years = (strategy_lin.index[-1]-strategy_lin.index[0]).days/365
cagr = (1+retorno_acum)**(1/years)-1
f'{cagr:.2%}, {retorno_acum:.2%}'
'55.67%, 6926.19%'

retorno_acum = (benchmark_lin+1).prod()-1
years = (benchmark_lin.index[-1]-benchmark_lin.index[0]).days/365
cagr = (1+retorno_acum)**(1/years)-1
f'{cagr:.2%}, {retorno_acum:.2%}'
'64.28%, 11683.92%'
```

Corroboramos en el cuadro que nos da lo mismo (igual ojo, acá hago una salvedad, a veces pasa, y de hecho van a ver que nos va a pasar que no nos coinciden los cálculos, con lo cual, luego de revisar que las fórmulas están bien y no fue un dedazo, tenemos que interpretar que es lo que hace la diferencia, muchas veces son criterios a tomar, ya veremos ejemplos muy claros con esta librería, pero resumiendo, lo importante no es que si o si nos de lo mismo, sino entender en todo caso donde radica la diferencia y cual de las formas de cálculo nos conviene asumir en el caso que hubieran diferencias)

Performance Metrics		
	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
Volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.83	0.39
Kurtosis	14.29	6.13
Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.99%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (cVaR)	-4.02%	-5.49%

En este caso nos da exactamente lo mismo, de hecho no hay mucho que discutir en como calcular un CAGR

Volatilidad

Para seguir un orden de dificultad y de cadenas de cálculo (estos datos los vamos a ir necesitando luego), no necesariamente el mismo orden que la librería, calculamos las volatilidades anualizadas, nada nuevo por ahora:

```

volatilidad = strategy_lin.std() * 252**0.5
volatilidad

0.4078614805975386

volatilidad_bench = benchmark_lin.std() * 252**0.5
volatilidad_bench

0.5548856810252318

```

Sharpe Ratio

Seguimos con uno de los ratios mas utilizados en finanzas para medir el riesgo, fue propuesto inicialmente por William Forsyth Sharpe (Economista que posteriormente obtuvo un premio nobel de economía, no por proponer este ratio obviamente, sino por sus aportes junto a Markowitz y Miller a la MPT)

La idea es super sencilla, se define como Sharpe Ratio " S_R " a la relación del retorno " R " menos el retorno del capital libre de riesgo " R_f " y la volatilidad del instrumento o cartera " σ ". Al usarse este ratio para tomar decisiones de inversión o cartera, hablamos de un "Retorno esperado" y no de un retorno realizado, aunque obviamente para estimar el retorno esperado vamos a basarnos generalmente en los retornos pasados

$$S_R = \frac{E[R - R_f]}{\sigma}$$

Bueno, a la hora de implementar el cálculo podemos tomar diferentes criterios, el mas usado es simplemente tomar una serie y calcular ambas partes (numerador y denominador con esa misma serie)
 Pero también se suele calcular agregados o ventanas móviles, vamos por partes, empezemos por lo mas usual que, de hecho, es lo que calcula nuestra librería quantstats

```
sharpe_st = strategy_lin.mean() / strategy_lin.std() * (252**0.5)
sharpe_bench = benchmark_lin.mean() / benchmark_lin.std() * (252**0.5)

sharpe_st, sharpe_bench

(1.2900444296797506, 1.1733267192132228)
```

Performance Metrics

	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
Volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.83	0.39
Kurtosis	14.29	6.13

Si se quisieran tener valores simétricos de un sharpe ratio tanto para los positivos como los negativos, se deberían usar retornos logarítmicos

Bien, pero dijimos que había otras formas de calcular el sharpe ratio, una de ellas es hacerlo tomando agregados de temporalidades más altas, por ejemplo anuales, eso sería usar el CAGR para los retornos y la volatilidad anualizada, es más realista que la media de los retornos diarios porque tiene en cuenta el efecto del interés compuesto digamos, pero de todos modos quiero hacer una salvedad, en realidad no importa tanto la forma de cálculo comparado con la homogeneidad, es decir, si para el método uso la misma forma de cálculo que para el benchmark, ambos tienen el mismo criterio de evaluación, lo que se dice "medir con la misma vara", después lo estricta que sea la vara, mientras sea la misma es menos relevante, a eso iba.

```
retorno_acum = (strategy_lin+1).prod()-1

years = (strategy_lin.index[-1]-strategy_lin.index[0]).days/365
cagr = (1+retorno_acum)**(1/years)-1

retorno_acum_benchmark = (benchmark_lin+1).prod()-1
years = (benchmark_lin.index[-1]-benchmark_lin.index[0]).days/365
cagr_benchmark = (1+retorno_acum_benchmark)**(1/years)-1

cagr, cagr_benchmark

(0.5566899805591667, 0.642762546612647)

sharpe_agg = cagr / volatilidad
sharpe_agg_benchmark = cagr_benchmark / volatilidad_bench

sharpe_agg, sharpe_agg_benchmark

(1.3648996216646558, 1.1583693156850792)
```

De hecho como ven, sigue dando el mismo resultado con un sharpe mejor en la estrategia que el Buy&Hold, pero mejorando en el primer caso y bajando en el segundo al tomar un criterio de composición.

Pero esas son "fotos" o mejor dicho datos únicos de toda una película, claramente, tanto estrategia como Buy&Hold tuvieron sus buenos y malos momentos, así que la otra forma de evaluar el Sharpe Ratio, es haciendo una ventana móvil del mismo, veamos la implementación

$$\overline{sharpe}_w = \frac{\sum_{i=1}^w R_i - F_{risk}}{\frac{1}{w} \sum_{i=1}^w \sum_{j=1}^n \frac{(R_j - \bar{R})^2}{n}}$$

Y en código Python que es mucho más fácil de leer:

```
w = 252
fr = 0

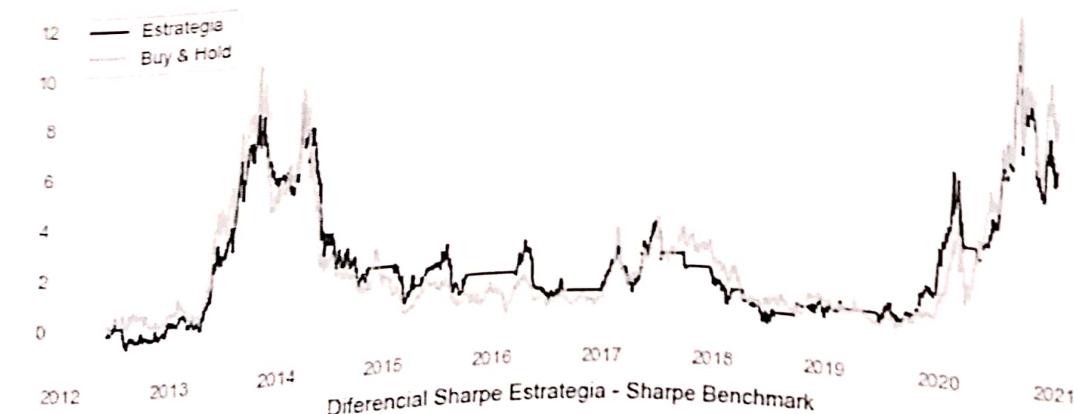
# calculamos numerador y denominador para la estrategia
roll_ret = strategy.pct_change(w)
roll_std = (strategy_lin.rolling(w).std() * (w**.5))

# calculamos numerador y denominador para el benchmark
roll_ret_benchmark = benchmark.pct_change(w)
roll_std_benchmark = (benchmark_lin.rolling(w).std() * (w**.5))

# calculamos las series
roll_sharpe = (roll_ret - fr) / roll_std
roll_sharpe_benchmark = (roll_ret_benchmark - fr) / roll_std_benchmark

fig, ax = plt.subplots(figsize=(12,10), nrows=2)
ax[0].plot(roll_sharpe, color='black', label='Estrategia')
ax[0].plot(roll_sharpe_benchmark, color='silver', label='Buy & Hold')
ax[0].legend()
ax[0].set_title(f'Rolling Sharpe Ventana de {w} velas diarias', fontsize=14)
ax[1].set_title('Diferencial Sharpe Estrategia - Sharpe Benchmark', fontsize=14)
ax[1].plot(roll_sharpe - roll_sharpe_benchmark)
```

Rolling Sharpe Ventana de 252 velas diarias



Sortino Ratio

Bien, hasta acá está bien esto, pero podría alguien decirme

"¿Juan, que tiene de malo que se dispare la volatilidad porque el activo sube un 25% en un día?"

Y la verdad, que tiene razón, si un día el activo en el que estoy comprado se dispara +25% por alguna buena noticia, la volatilidad se va a disparar también porque el desvío estándar de los retornos lo hace, pero no tiene nada de malo, lo que nos jode como inversores es "la volatilidad mala" que uno se tienta a decir que es la de los días que el activo baja, pero esto asumiendo una cartera Long, es decir comprada, la definición sería que lo que nos jode es la volatilidad en el sentido opuesto a nuestra posición.

Esta es la forma en que lo implementa el autor de la librería, yo personalmente prefiero partir de entrada al subconjunto de los retornos negativos y luego hacer todo el ratio con la media de esos retornos y ese q, pero es un detalle, como dije antes, lo importante es medir siempre con la misma vara y por supuesto tener en cuenta las implicancias de incorporar un criterio u otro en el cálculo

```
def sortino(returns):
    semi_desv = (((returns[returns < 0] ) **2).sum() / len(returns)) **.5
    return returns.mean() / semi_desv * (252**0.5)

sortino(strategy_lin), sortino(benchmark_lin)
(2.0770118004271607, 1.8239226035350857)
```

Performance Metrics

	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
calmar	1.46	1.06
skew	0.83	0.39
kurtosis	14.29	6.13

Y acá aplica lo mismo que dijimos para el ratio Sharpe, podríamos tomar un agregado anualizado, o también podríamos graficar una película con una ventana móvil, no lo hago pero se los dejo como ejercicio

Máximo Drawdown

Este es mi indicador favorito de análisis de riesgo.

Pero veamos primero un esquemita:

Max Drawdown
Es la máxima caída
desde ATH previo



Longitud de Drawdown

Es la cantidad de días/años que dura
cada Drawdown hasta recuperar ATH

Ahora el punto es por que a un inversor podría preocuparle el Drawdown más que la volatilidad. Eso es super relativo a la aversión al riesgo de cada uno, en mi caso, le tengo mas aversión a los drawdowns y sobre todo a la longitud, aunque también en menor medida a la profundidad, pero digamos que por mi ansiedad, me cuesta mucho "bancar" un Drawdown de muchos años, ahora, si es un Drawdown de menos del 30% es otra cosa, pero drawdowns del 50% o mas que duren mas de 4 años, en mi caso es devastador, sin embargo puedo soportar una volatilidad diaria furiosa al estilo bitcoin y cia, pero me cuesta mucho sobrellevar un Drawdown profundo y largo

Ahora lo interesante viene cuando combinemos ese máximo Drawdown contra un retorno, como es el caso del Calmar Ratio, pero empieceremos con la implementación del máximo Drawdown y luego seguimos con los ratios.

```
# Calculo las series de drawdowns
dd_strategy = (strategy / strategy.cummax()-1)
dd_benchmark = (benchmark / benchmark.cummax()-1)

# Veo solo los mínimos
dd_strategy.min(), dd_benchmark.min()

(-0.3811355084988274, -0.6062653645917144)
```

Performance Metrics

	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
Volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.83	0.39
Kurtosis	14.29	6.13

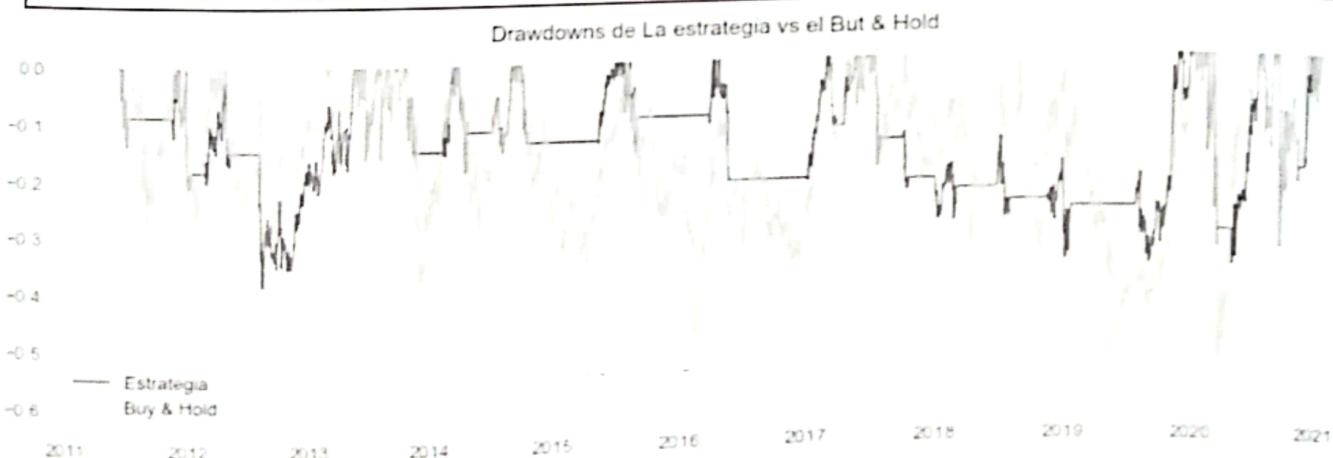
Como ven, si bien tenemos menor CAGR o sea menor rendimiento que el Buy&Hold, en realidad ya sabíamos que en términos comparados con la volatilidad, teníamos mejor recompensa por unidad de volatilidad, bueno, ahora vemos que los drawdowns también nos indican que bajamos mucho el riesgo con la estrategia, lo cual es un buen indicio (recuerden siempre el trilema, no se pueden todas, si queremos menos riesgo varmos a tener que sacrificar rentabilidad, la gracia es encontrar buenas relaciones para luego combinar estrategias que me aporten upside con estrategias que me aporten bajo riesgo)

Y podemos graficar la película entera en lugar de ver solo los máximos drawdowns

```
fig, ax = plt.subplots(figsize=(15,5))

ax.plot(dd_strategy, color='k', label='Estrategia', alpha=0.7)
ax.plot(dd_benchmark, color='silver', label='Buy & Hold', alpha=0.7)

ax.set_title('Drawdowns de La estrategia vs el Buy & Hold', fontsize=14)
ax.legend()
```



Duración de Drawdowns

Hablamos dicho que la profundidad del Drawdown sola, no es todo lo que nos importa, sino la combinación de la profundidad con la duración, ya que una gran profundidad del 60% es tolerable si dura pocos días (ejemplo marzo de 2020 x covid, muchos activos tuvieron una baja de más del 50% pero a los pocos días recuperó bastante y en unos meses ya había vuelto a su valor pre crisis)

Es decir, es mucho más tolerable un Drawdown del 60% que dure un par de meses que uno del 55% pero que dure 6 años, al menos en mi caso. Con lo cual es de vital importancia calcular la longitud del mayor o los mayores drawdowns, muchos hacen una especie de "puntaje" o ratios combinando ambos (profundidad/duración) lo importante es considerarlos, después cada uno, según su aversión implementará la métrica o combinación que más le sirva.

Entonces, calculemos esas duraciones, en este caso les muestro mi solución para calcular el top 10 de drawdowns, les pongo en comentario la descripción de lo que hace cada línea.

```
# Busco ATHs
maximos_mascara = strategy.cummax() == strategy

# Lo guardo los valores de las fechas de Los ATHs (los precios no me importan aca)
dates = maximos_mascara.loc[maximos_mascara].index

# Calculo las diferencias de días entre fechas de ATHs
dd_days = pd.Series(dates).diff()

# Ordenos de mayor a menor
top10 = dd_days.sort_values(ascending=False).head(10).reset_index(drop=True)
top10
```

0	868 days
1	523 days
2	309 days
3	277 days
4	242 days
5	164 days
6	160 days
7	141 days
8	133 days
9	98 days

Hago lo mismo para el benchmark y comparamos:

```
# Busco ATHs
maximos_mascara = benchmark.cummax() == benchmark

# Lo guardo los valores de las fechas de Los ATHs (los precios no me importan aca)
dates = maximos_mascara.loc[maximos_mascara].index

# Calculo las diferencias de días entre fechas de ATHs
dd_days = pd.Series(dates).diff()

# Ordenos de mayor a menor
top10_benchmark = dd_days.sort_values(ascending=False).head(10).reset_index(drop=True)
top10, top10_benchmark
```

Pego las dos tablas una al lado de la otra en la siguiente página para apreciarlo mejor

Duración Drawdowns

Estrategia	Benchmark
0 868 days	0 942 days
1 523 days	1 821 days
2 309 days	2 300 days
3 277 days	3 160 days
4 242 days	4 156 days
5 164 days	5 133 days
6 160 days	6 110 days
7 141 days	7 93 days
8 133 days	8 87 days
9 98 days	9 80 days

Como ven tornando los principales (se suele evaluar los de más de 1 año, pero es algo muy arbitrario también), no hay gran diferencia, tenemos una leve mejoría en la duración con la estrategia, pero siguen siendo drawdowns bastante largos de más de 2 años el primero, y el segundo al menos lo redujo a año y medio aproximadamente cuando el segundo del buy & hold era de más de 2 años.

Igualmente no hay que perder de vista que ya habíamos mejorado la profundidad de los drawdowns un montón, con lo cual si al menos logramos reducir un poquito la duración ya es bastante positivo en este aspecto.

Performance Metrics

	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
Volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.83	0.39
Kurtosis	14.29	6.13

Correlación

Otra métrica importante es la correlación de mi estrategia contra el benchmark, generalmente es alta pero hay casos (y esto es el mejor escenario) en los que la correlación o es baja o negativa.

¿Por qué digo que sería bueno una correlación baja?

Bueno, principalmente uno de los mayores problemas en el armado de carteras es la alta correlación de todos los activos, sobre todo en los momentos de crisis o gran volatilidad. Este problema de la alta correlación hace que las diversificaciones no tengan el efecto deseado (dicho en forma muy resumida)

Con lo cual, si logramos estrategias que "se despeguen" lo suficiente de lo que hace el activo que tradeamos en general, tendremos una estrategia que, mas allá de si nos convencen o no los ratios de riesgo propios, nos servirá bastante para diversificar el portfolio en general

Bien, Entendido esto, de la necesidad de despegarse del activo que tradeamos, la pregunta es ¿Cómo medimos ese "despegue" objetivamente?

La respuesta es con el R^2 o coeficiente de correlación, entre el benchmark (o activo que tradeamos) y nuestro payoff, esto ya lo vimos en el t[2] si mal no recuerdo, pero repasando usamos el coeficiente de Pearson generalmente, al que llamamos ρ , que es el cociente entre la covarianza de dos distribuciones y el producto de sus desvíos:

$$\rho_{\hat{\alpha}, \hat{\beta}} = \frac{\text{cov}(\hat{\alpha}, \hat{\beta})}{\sigma_{\hat{\alpha}} \sigma_{\hat{\beta}}}$$

Afortunadamente Python y específicamente pandas, lo hace todo muy fácil para nosotros:

```
rho = strategy_lin.corr(benchmark_lin)
rho
0.7340371518062246

r2 = round(rho**2,3)
r2
0.539
```

De todos modos, si lo queremos calcular con la fórmula que puse arriba es casi igual de fácil:

```
# calculando con la covarianza y los desvíos

rho = strategy_lin.cov(benchmark_lin) / (strategy_lin.std() * benchmark_lin.std())
r2 = round(rho**2,4)
r2
0.539
```

Performance Metrics

	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.83	0.39
Kurtosis	14.29	6.13

Como ven es ese 0.54 que figuraba en la tabla que teníamos, pero quiero que presten atención a ese ratio "Time in Market" y hagan un ejercicio mental de como incide en el R2 en una estrategia de momentum simple como esta solo long.

Bueno, si leyeron el cuadrito ese que puse a la derecha de la tabla, y si hicieron el ejercicio mental habrán caido que "a mas tiempo dentro permanezca la estrategia" => "mas alta será la correlación"

Parece una obviedad, no lo es tanto en realidad, es decir, obvio que si la estrategia me deja comprado el 90% de los días es de esperar que la misma tenga una correlación cercana a 0.9, de la misma forma que si solo permanece comprado el 10% de los días, es de esperar que la correlación sea baja, pero también depende de "que días?" esté comprado con la estrategia, igualmente el concepto es que si tenemos muchas estrategias de bajo "Time in market" lo que lograríamos es "despegarnos fuerte del benchmark" y esto en portafolios y gestión del riesgo es el santo grial

Momentos clave en la correlación

Este es un tema mas de Risk Management y portfolio más que de backtesting, pero no podía seguir adelante con los ratios sin siquiera mencionarlo, acabo de decir en la página anterior que es importante poder "despegarte del benchmark" para que la estrategia sea útil para diversificar (además de las virtudes propias)

Ahora la siguiente pregunta es ¿alcanza solo con saber el R^2 para entender que tanto se despegó del benchmark?

Si planteo la pregunta, es casi obvio que la respuesta es NO, caso contrario ¿para que carajo lo preguntaría? Bueno, en realidad en términos cuantitativos es un gran indicador el R^2 , pero en términos prácticos no sirve mucho lamentablemente, lo voy a explicar a lo almacenero porque creo que se entiende mucho mejor así:

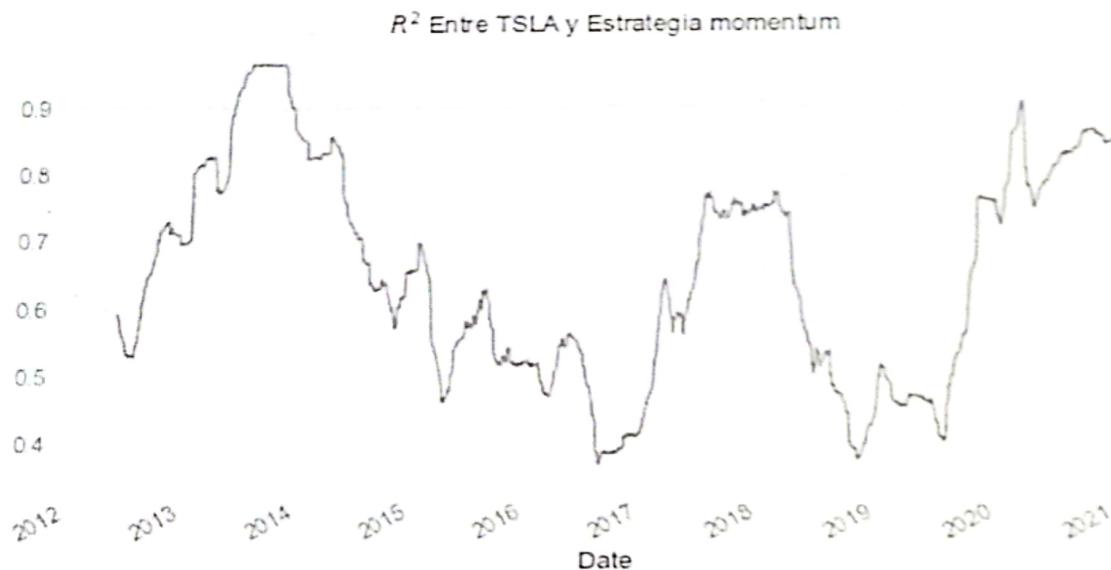
No es lo mismo que logre despegarme "en las buenas" que "en las malas"

Es decir, el $R^2=0.54$ me dice que mas o menos están pegados la mitad del tiempo, pero claramente en diversificación (sobre todo en estrategias Long como esta), lo que me interesa es despegarme cuando todo se hunde, es decir "en las malas" o en los drawdowns o al menos estar mas despegado ahí, si logro eso, será una muy buena estrategia para diversificar

La siguiente pregunta es ¿Cómo hago entonces para ver que tan en las malas o en las buenas se logró despegar? ¿Qué métrica uso? En fin, veamos un toque este punto

Lo mas directo es hacer primero un gráfico dinámico de correlación en el tiempo, eso con Python es tan sencillo como 1 (UNA) línea de código (amo Python)

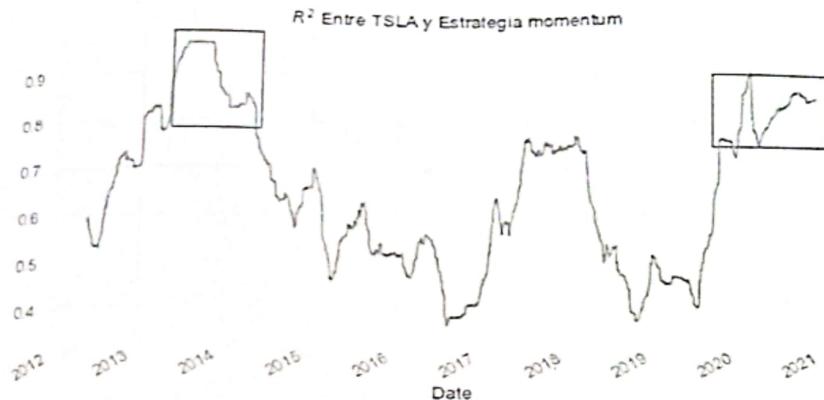
```
serie = strategy_lin.rolling(250).corr(benchmark_lin)
serie.plot(figsize=(10,5), title=r'$R^2$ Entre TSLA y Estrategia momentum')
```



Es decir un `rolling()` y como función en lugar de `mean()` o algo así, usamos `corr()` y le pasamos como argumento la serie contra la que queremos correlacionar, es decir, una serie de retornos contra la otra, ahora veamos lo que nos interesaba chequear, que es ¿Cuándo se despegó mas y cuándo se despegó menos del benchmark? Recordemos que nuestro benchmark es la acción de TSLA que es la que estamos tardeando

Si vemos en donde se despega menos (R^2 más alto) es en las buenas o neutras (lo cual es bueno, porque lo que menos queremos es despegarnos ahí)

Justamente da muy alto el R^2 a fines de 2013 y fines de 2020, dos momentos hiper bullish para TSLA, lo cual es bueno



Acá se confunde un poco, pero vemos R^2 y precio, superpuestos en el mismo gráfico

```
fig, ax = plt.subplots(figsize=(12,6))
serie = strategy_lin.rolling(250).corr(benchmark_lin)
ax.plot(serie, color='k', label=r'$R^2$ Entre TSLA y Estrategia momentum')
ax.legend(loc = 'upper left')

ax2 = ax.twinx()
ax2.plot(data.Close, color='silver', label='TSLA Price')
ax2.legend(loc='upper right')
ax2.set_yscale('log')
```



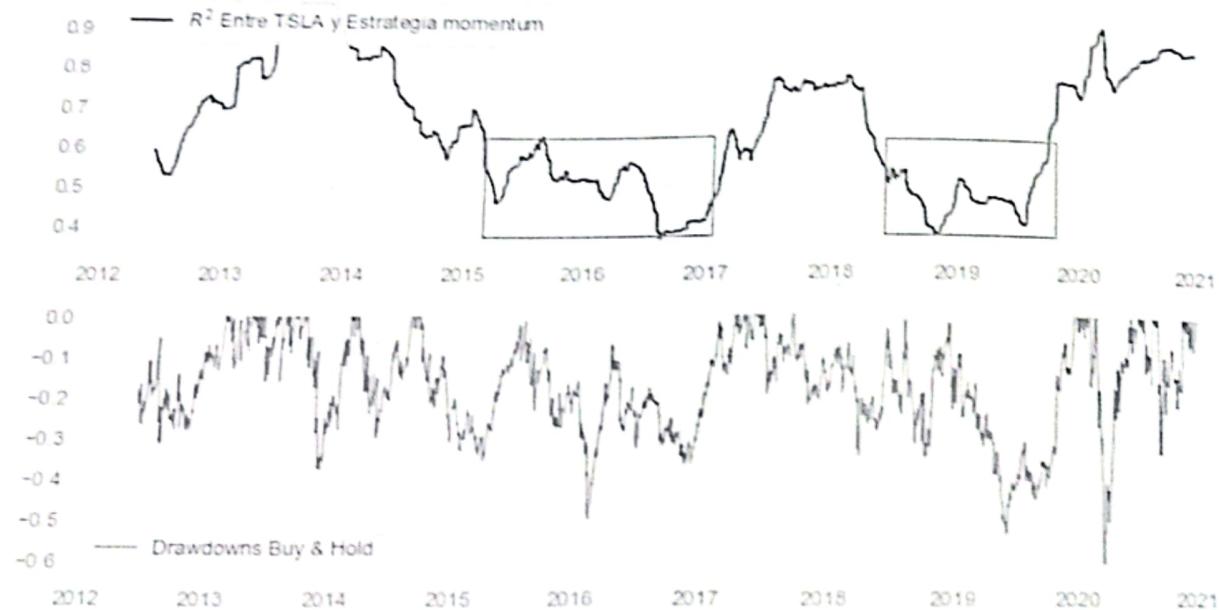
Ahora eso es cuando "se despega menos", y lo que queremos ver también es cuando "se despega mas" que pasa con el precio, al ser tan bullish TSLA se ve mejor con el gráfico de los drawdowns que con el del precio esto

Miren que claro que se ve:

```
fig, ax = plt.subplots(figsize=(12,9), nrows=2)

serie = strategy_lin.rolling(250).corr(benchmark_lin)
ax[0].plot(serie, color='k', label=r'$R^2$ Entre TSLA y Estrategia momentum')
ax[0].legend(loc = 'upper left')

ax[1].plot(dd_benchmark.iloc[250:], 'k', lw=1, label='Drawdowns Buy & Hold', alpha=0.7)
ax[1].legend(loc='lower left')
```



Y eso que el Rolling aplicado al R^2 "lo retrasa", miren como se ve clarísimo si le aplico el mismo Rolling de 1 año en días de negociación a la serie de drawdowns:

```
fig, ax = plt.subplots(figsize=(12,9), nrows=2)

serie = strategy_lin.rolling(250).corr(benchmark_lin)
ax[0].plot(serie, color='k', label=r'$R^2$ Entre TSLA y Estrategia momentum')
ax[0].legend(loc = 'upper left')

ax[1].plot(dd_benchmark.rolling(250).mean(), 'k', lw=1, label='Drawdowns Buy & Hold', alpha=0.7)
ax[1].legend(loc='lower left')
```



Conclusión respecto a este tema, al ser un R^2 bajo, pero especialmente más bajo "en las malas" me sirve la estrategia para diversificar, es decir que a futuro puedo usarla para invertir una parte de mi portafolio en esta estrategia de TSLA que seguramente tendrá buenos índices de portfolio y risk management con otros payoffs "más pegados" al mercado o este activo en particular

Calmar Ratio

Haciendo un breve repaso el nombre es un acrónimo de "California Managed Accounts Reports" que es la compañía que, como se imaginarán, comenzó a publicar reportes con este ratio que luego se popularizó.

La idea es muy simple:

Comprar el dolor de la máxima caída contra el rendimiento anualizado compuesto o CAGR

```
calmar_strategy = cagr / dd_strategy.abs().max()
calmar_strategy
1.4606090698602003

calmar_benchmark = cagr_benchmark / dd_benchmark.abs().max()
calmar_benchmark
1.0602000116657024
```

Performance Metrics

	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
Volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.63	0.59
Kurtosis	14.29	6.13

Es un ratio muy simple, fácil de calcular, y bastante informativo para quienes (me incluyo) medimos el riesgo con "parámetros de drawdowns"

Única observación que tengo al Calmar ratio es que solo sirve a nivel comparativo y no absoluto como el sharpe, es decir, un sharpe > 1 , es un buen indicador, pero un Calmar > 1 no me dice nada en sí

¿Por qué?

Fácil, porque es relativo al tiempo. Imaginen una estrategia durante 50 años en una acción, es obvio que en ese plazo va a tener un Drawdown del 50% para empezar a hablar, y el CAGR compuesto en 50 años difícilmente sea mayor al 20% como super exitosa la empresa. Es decir que en 50 años un Calmar de 11 sería algo increíblemente bueno por no decir imposible de conseguir, digamos que en ese plazo un Calmar excelente sería 0.4 por decir algo (20%/50%)

¿pero en 1 año?

Ahi es muy probable encontrar varios con CAGR > 50% y DD < 10%, claaaa, solo un año, pero en ese plazo podria darse un Calmar Ratio = 5 tranquilamente, deben haber miles de ejemplos en el SP500 en 2021

Así que resumiendo, es un ratio muy interesante por lo sencillo y práctico pero solo a nivel comparativo Estrategia/Benchmark en el mismo plazo, no puedo comparar un Calmar en un plazo de X años con otro de un plazo diferente.

Asimetria y Kurtosis

Este tema también lo hemos visto en otros tomos, sobre todo la kurtosis se asocia mucho a las colas pero también puede darse una kurtosis alta por muchos valores cercanos a la media, por eso se habla de 2 tipos de kurtosis, la kurtosis de cola y la kurtosis de pico (centro de la distribución)

```
strategy_lin.skew(), strategy_lin.kurtosis()
```

```
(0.8264223800015249, 14.281383082941225)
```

```
benchmark_lin.skew(), benchmark_lin.kurtosis()
```

```
(0.38720525603286376, 6.122115385370812)
```

Performance Metrics		
	Strategy	Benchmark
Start Period	2011-05-25	2011-05-25
End Period	2020-12-30	2020-12-30
Risk-Free Rate	0.0%	0.0%
Time in Market	52.0%	100.0%
Cumulative Return	6,926.19%	11,683.92%
CAGR%	55.67%	64.28%
Sharpe	1.29	1.17
Sortino	2.08	1.82
Max Drawdown	-38.11%	-60.63%
Longest DD Days	865	941
Volatility (ann.)	40.78%	55.47%
R^2	0.54	0.54
Calmar	1.46	1.06
Skew	0.83	0.39
Kurtosis	14.29	6.13

En este caso como en muchos del tipo similar a este es obvio que la kurtosis va a dar super alta

¿Por qué es obvio?

Y porque todos los días que no da señal de compra, está afuera "con payoff 0%" es decir que habrá una moda con una cantidad de datos enorme en 0% en la estrategia que no ocurre en el buy&Hold con lo cual se dispara la kurtosis de centro de distribución y no es para nada malo..

Lo que si sería malo es una kurtosis de colas, es decir que se dispare la kurtosis por muchos valores muy extremos, ya que sabemos que las cosas extremas no son de agrado en general de los inversores (los extremos malos obvio, pero generalmente cuando hay mucho extremo negativo también lo hay del lado positivo)

Pero bueno no me detengo en kurtosis y asimetría porque lo vimos en el t[6] y tomó anteriores así que avanza porque queda todavía todas estas métricas para repasar:

expected Daily %	0.18%	0.2%	MTD	22.41%	22.41%
expected Monthly %	3.73%	4.2%	3M	35.76%	64.95%
expected Yearly %	52.90%	61.11%	6M	183.26%	244.17%
Kelly Criterion	15.43%	10.71%	YTD	474.97%	730.42%
Risk of Ruin	0.0%	0.0%	1Y	480.0%	737.69%
Daily Value-at-Risk	-4.02%	-5.49%	3Y (ann.)	116.49%	123.78%
Expected Shortfall (cVaR)	-4.02%	-5.49%	5Y (ann.)	63.51%	70.96%
			10Y (ann.)	55.67%	64.28%
			All-time (ann.)	55.67%	64.28%
Payoff Ratio	1.17	1.17	Best Day	24.4%	24.4%
Profit Factor	1.4	1.24	Worst Day	-21.06%	-21.06%
Common Sense Ratio	1.65	1.44	Best Month	81.07%	81.07%
CPC Index	0.89	0.76	Worst Month	-23.75%	-22.43%
Tail Ratio	1.18	1.15	Best Year	474.97%	730.42%
Outlier Win Ratio	10.74	4.1	Worst Year	-12.63%	-10.97%
Outlier Loss Ratio	3.62	3.67			
Avg. Drawdown	-7.47%	-9.03%			
Avg. Drawdown Days	51	46			
Recovery Factor	181.73	192.72			
Ulcer Index	inf	1.01			
Avg. Up Month	16.45%	20.48%			
Avg. Down Month	-8.8%	-8.81%			
Win Days %	54.39%	51.85%			
Win Month %	64.0%	55.17%			
Win Quarter %	63.64%	53.85%			
Win Year %	80.0%	80.0%			
Beta	0.54	-			
Alpha	0.17	-			

Arranquemos por lo que es más de performance, y resampleos

Está claro que una de las cosas a saber es como resultó la estrategia el último mes, los últimos 3 meses, año etc. porque uno tiene fresco lo sucedido en la coyuntura en esos momentos recientes, esto lo podemos catalogar como métricas de performance corrientes, no pierdo espacio en poner esto porque a esta altura tienen las herramientas suficientes para hacerlo con pandas.

Lo mismo las métricas del mejor o peor año, mes, semana, etc, es todo lo que puse en la columna de la derecha, realmente es fácil de entender y de calcular así que se los dejo de ejercitación, vamos a concentrarnos en la columna de la izquierda no porque sea más importante que lo otro, sino que seguramente habrá cosas nuevas que no vieron hasta acá, mientras que lo otro es bastante obvio.

Esperanza Matemática

Cada vez que vean el "expected" seguido de algo, se está refiriendo a lo que se conoce como "esperanza matemática" es un término muy importante en estadística porque se refiere, no a un promedio, o un valor concreto, sino a un valor esperado, que probablemente no suceda nunca puntualmente, pero es "el esperado"

Pongo un ejemplo, si tiro 60 veces un dado, tengo una esperanza de que salga 10 veces el "5" por decir algo Ahora puede ser que salga solo 3 o 4 o quizás hasta ninguna (sería raro pero puede pasar), es decir que la esperanza conceptualmente no nos da un valor determinístico o algo como "el promedio" o ese tipo de cosas, sino que nos da justamente "una esperanza" de algo que a futuro puede suceder

Perdón el rollo con este concepto pero es muy importante que conceptualmente se entienda la diferencia entre promedio y esperanza.

Bueno, ahora a calcular esperanzas, por ejemplo, calculemos las esperanzas de retornos diarios, mensuales y anuales

Si dijimos que es el valor esperado a futuro, claramente no lo vamos a calcular como un promedio simple, sino que vamos a inducir cual sería el valor mas esperable de retorno futuro (que como dijimos no es el promedio de los valores pasados)

Bueno, cada cálculo del "expected value" sea lo que sea el value, tiene su lógica, pero conceptualmente es una simulación, es decir sabiendo los datos pasados, debemos inferir cual sería el valor esperado futuro

En tema retornos podríamos tomar un interés compuesto de los retornos en "n" períodos y asumir que si todos los días tuviera una tasa esperada " $E[R]$ " entonces asumiendo que queremos simular todos los r_i retornos diarios, tenemos:

$$(E[R] + 1)^n = \prod_{i=1}^n 1 + r_i$$

Ya sé, ya sé, fórmulas no gustan, va mejor código Python

```
q = len(strategy_lin)
exp_daily = strategy_lin.add(1).prod() ** (1/q) - 1
exp_daily
0.0017608494241054817

months = len(strategy.resample('M').first())
exp_monthly = strategy_lin.add(1).prod() ** (1/months) - 1
exp_monthly
0.03733731759445691

years = len(strategy.resample('Y').first())
exp_yearly = strategy_lin.add(1).prod() ** (1/years) - 1
exp_yearly
0.5299316155836202
```

O sea que si esperamos que todos los días haya un retorno del 0.176%, todos los meses uno de 3.733% y todos los años un retorno del 52.9931%

Ahora quizá no haya ni un solo día, mes o año con ese retorno, ni tampoco es un promedio, sino un valor esperado, el más esperado si quieren.

Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.99%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (cVaR)	-4.02%	-5.49%
Payoff Ratio	1.17	1.17
Profit Factor	1.4	1.24
Common Sense Ratio	1.65	1.44
CPC Index	0.89	0.76
Tail Ratio	1.18	1.15
Outlier Win Ratio	10.74	4.1
Outlier Loss Ratio	3.62	3.67

Obviamente que remplazan en el código donde diga strategy por benchmark y obtienen las esperanzas del benchmark

Criterio de Kelly

Este indicador viene del mundo de las apuestas deportivas pero aplica bien a modelos de trading que tengan altos porcentajes de ganancias o pérdidas, en nuestro caso el payoff es la variación diaria, e incluso con un activo muy volátil como TSLA, esta variación diaria por mas alta que sea la varianza, no tiene implícito un riesgo de ruina considerable ¿Qué es eso?

Bueno, imaginemos que tenemos velas o días con cierta probabilidad palpable de perder un 30%... esto no es así en la bolsa, al menos no con TSLA, pero imaginémoslo un momento.. En esa situación si tenemos una racha mala de 10 días con esa pérdida.. prácticamente fundimos, o quedamos con un valor de portafolio de apenas un 2.82% del valor inicial, lo que equivaldría a fundir digamos

```
0.7**10  
0.02824752489999998
```

Bueno, lo que viene a plantear el criterio de Kelly es un "tamaño de apuesta" óptimo para minimizar esas chances de quebrar, pero maximizando en lo posible el potencial upside de estar invertido..

Vean la paradoja, si invertimos el 100% del capital, esa racha de 10 velas seguidas de las malas (imaginando que sea probable esa racha en TSLA) nos deja out, pero si por el contrario invertimos solo el 10% del capital, la pérdida es de apenas el 9.71% en lugar del 97.18% pero claro, por otro lado estamos acotando muchísimo el upside, y la verdad no era necesario bajar tanto la cantidad invertida, fue exagerado solo invertir el 10%, porque invirtiendo el 50% tampoco fundiamos con esa racha de 10 ruedas, en fin, el punto creo que se entiende, la idea es reducir lo mas posible la posibilidad de quiebra pero maximizando el upside o % invertido.

Fórmula:

$$f^* = p - \frac{q}{b} = p + \frac{p-1}{b}$$

- f^* es la fracción o porcentaje óptimo a invertir que decíamos
- p es la probabilidad que el activo suba
- q es la probabilidad que baje o $1-p$
- b es el ratio de lo que se gana cuando se gana vs lo que se pierde cuando se pierde

Implementación en Python:

```
perdedores = strategy_lin[strategy_lin < 0]
ganadores = strategy_lin[strategy_lin > 0]
totales = strategy_lin[strategy_lin != 0]

b = -ganadores.mean() / perdedores.mean()
prob_ganador = len(ganadores) / len(totales)
prob_perdedor = 1 - prob_ganador

kelly = prob_ganador - prob_perdedor / b
kelly

0.15346761755753274
```

O sea que ese 15.34% es la fracción óptima a invertir según este criterio de Kelly en esta estrategia de TSLA. Hay un temita ahí de como contabilizar los 0% si como ganadores o perdedores, pero es irrelevante a lo conceptual ya que incide en unos decimales apenas.

Y en el caso del benchmark me da menos (un 10.1%)

```
perdedores = benchmark_lin[benchmark_lin < 0]
ganadores = benchmark_lin[benchmark_lin > 0]
totales = benchmark_lin[benchmark_lin != 0]

b = -ganadores.mean() / perdedores.mean()
prob_ganador = len(ganadores) / len(totales)
prob_perdedor = 1 - prob_ganador

kelly = prob_ganador - prob_perdedor / b
kelly
```

0.10108575193315811

Claramente es mejor que sea mas alto el "f" ya que me permitiría mas upside, o visto del otro lado, un "f" de Kelly mas bajo significa un método "más arriesgado" (siempre tomando al riesgo como una contraparte del retorno)

Es decir que el indicador del criterio de Kelly me vuelve a marcar lo mismo que el **Calmar Ratio** y lo mismo que los drawdowns, es decir que la estrategia es menos riesgosa que el benchmark.

Riesgo de Ruina

Siguiendo con esta tanda de indicadores de bancarrota, viene lo que se conoce como el "riesgo de ruina", este indicador es realmente muy relevante cuando se trata de payoffs de % muy altos, es decir que cada variación (ejemplo cada vela) puede ser +/- 30% por poner un ejemplo, en esos casos, el problema es que si se da una racha "atípica" pero posible, por ejemplo de 5 trades negativos seguidos de -30%, en ese caso perderíamos el 83% del capital, mientras que si se da la racha de 10 seguidas perderíamos el 97% del capital, con lo cual podría considerarse una ruina, es decir la bancarrota, ya que recuperar la posición inicial luego de un -97% es casi imposible.

Bueno, lo que trataremos de ponderar en este indicador es "¿Qué tan probable es que terminemos en este tipo de situaciones dadas las variaciones de nuestro payoff y dado el valor que consideramos ruina?"

Ahora para poder calcular esto de un modo objetivo vamos a tener que "asumir algo"

Básicamente hay dos formas típicas de calcular esto,

- 1- Asumiendo una distribución (modelando), generalmente se usa la distribución normal pero como siempre digo, ojo con estas cosas porque es una simplificación bastante naif en la bolsa.
- 2- Simulando, es decir, asumiendo que no hay un modelo de distribución teórica, sino que simulamos un cálculo con los datos empíricos pasados, sin importar si modelan o no algún tipo de distribución.

Obviamente si es por "realismo" uno elegiría el segundo método, si es por simplicidad, elegiría el primero. Acá loes voy a mostrar la forma de calcularlo de las dos maneras obviamente, según el caso apliquen lo que más crean conveniente.

Riesgo de Ruina Asumiendo normalidad

Partimos de la siguiente formulación:

$$P = \left(\frac{2}{1 + \frac{\mu}{r}} - 1 \right)^{\frac{s_0}{r}}$$

$$r = \sqrt{\mu^2 + \sigma^2}$$

Llamamos S_0 al valor inicial del subyacente, y obviamente " μ " a la media de variación de S y " σ " al desvio estándar en las variaciones de S . Si tomamos variaciones porcentuales (sobre 1) podemos tomar como $S_0=1$, pero si tomamos variaciones en términos de USD (ejemplo una media de variación de USD 1.15 por decir algo) tendríamos que tomar el S_0 como el valor en USD Inicial del portafolio para el cual vamos a calcular el riesgo de ruina. Obviamente que por una cuestión de normalización se suele usar una base 1.

Aplicando entonces la formulita calculemos en Python el riesgo de ruina de nuestra estrategia (el swing trading de TSLA) y del benchmark (TSLA en modo buy&Hold)

```

mu = strategy_lin.mean()
sigma = strategy_lin.std()
s_0 = 1

r = (mu**2 + sigma**2)**0.5
riesgo_ruina = (2/(1+mu/r)-1)**(s_0/r)
f"Riesgo Ruina estrategia {riesgo_ruina:.4%}"

'Riesgo Ruina estrategia 0.1840%'

mu = benchmark_lin.mean()
sigma = benchmark_lin.std()
s_0 = 1

r = (mu**2 + sigma**2)**0.5
riesgo_ruina = (2/(1+mu/r)-1)**(s_0/r)
f"Riesgo Ruina benchmark {riesgo_ruina:.4%}"

'Riesgo Ruina benchmark 1.4791%'
```

Ahora antes de seguir con la otra forma de calcular este riesgo es importante interpretar bien este resultado. Por un lado, la estrategia marca ser menos riesgosa que el benchmark de conducirme a una ruina, hasta acá nada nuevo porque es lo mismo que me vienen marcando todos los otros indicadores.

El punto es entender en donde estamos yendo a la banquina asumiendo normalidad acá. Sabiendo que ni TSLA ni mi método de swing sobre TSLA tendrán una distribución normal

¿el verdadero riesgo de ruina debería ser más alto o más bajo que esto?

Háganse la pregunta en serio, no sigan avanzando, gran parte de un buen backtest es entender realmente los riesgos y obviamente poder calcularlos primero y simular parametrizaciones después para minimizarlos, pero si arrancamos sin entender bien lo conceptual estamos al horno

Asumo que se tomaron ya el tiempo para responderse esa pregunta que dejé abierta

La respuesta es más bajo ¿por qué? Porque estamos asumiendo que una variación muy fuera de escala (por ejemplo, si fuera el Merval la variación de -50% del 12/8/19) es parte de una distribución normal, y en realidad ese evento del orden de 17 sigmas, no es parte de un comportamiento normal, pero el modelado calcula el sigma de nuestra normal como una normal típica donde los eventos de cola en 5000 datos son eventos 2 o 3 sigmas a lo mucho. O sea, al forzar a meter nuestros datos empíricos a una normal típica estamos exagerando las colas o suponiéndolas parte de un comportamiento normal y por ende les asignamos sin querer mas probabilidad de ocurrencia, por eso el riesgo de quiebra da más alto generalmente modelando normalidad

Ahora vemos cómo funciona el otro método de calcular el verdadero riesgo de ruina con los datos empíricos

Riesgo de Ruina con datos empíricos

Arrancamos por llamar L_{MAX} a nuestra pérdida máxima tolerable (arbitriariamente lo fijamos por ejemplo en un 95% que sería lo que consideramos "quiebra" y por lo tanto el seteo que ponemos para calcular la probabilidad de que se llegue a ese valor tomando en una simulación aleatoriamente valores de la distribución real)

Luego definimos la inecuación dada porque nuestro total "1" menos esa pérdida máxima deberá ser siempre mayor o igual al resultado de una operación de pérdida media (dada por $1 - \text{la pérdida media}$) elevado a un número de pérdidas de la racha negativa "U" que es lo que queremos calcular

$$1 - L_{MAX} \geq (1 - \overline{\text{loss}})^U$$

En otras palabras, lo primero que calculo es la cantidad "U" de velas negativas que hacen que mi portafolio caiga por debajo de mi pérdida máxima seteada con L_{MAX}

Esto con las propiedades de los logaritmos se despeja muy fácil:

$$U = \frac{\log(1 - L_{MAX})}{\log(1 - \overline{\text{loss}})}$$

Y más fácil todavía es calcularlo con Python:

```
trade_perdedor = -strategy_lin[strategy_lin < 0].mean()
trade_perdedor
0.02231369130757361

# U = racha max tolerable de trades perdedores dada una perdida max tolerable.

max_perdida = 0.95
U = int(np.log(1-max_perdida) / np.log(1-trade_perdedor))
U
132
```

O sea que con nuestra estrategia tenemos una pérdida media del 2.2313% (el promedio de las velas perdedoras) Y para llegar entonces a una pérdida de portafolio del 95% necesitamos 132 velas promedio de esas. Obviamente ni con GGAL vamos a encontrar un activo que tenga 132 velas rojas seguidas, así que mucho de que preocuparse por ahora no tenemos, pero claro, seteamos una ruina asumiendo pérdida del 95%, pero que pasa si seteamos eso en un 10%? De esa forma calculamos la cantidad de "velas rojas necesarias" para cumplir la condición de "no-ruina" o no caer más del 10%

```
# U = racha max tolerable de trades perdedores dada una perdida max tolerable.

max_perdida = 0.1
U = int(np.log(1-max_perdida) / np.log(1-trade_perdedor))
U
4
```

Ahora es mucho mas probable que se de este escenario ¿o no?

Chequeamos los cálculos:

```
(1-trade_perdedor)**4, (1-trade_perdedor)**5, (1-trade_perdedor)**132  
(0.9136884475727733, 0.8933006856023382, 0.05085555718539359)
```

Efectivamente a la quinta "vela roja" seguida con ese promedio de pérdida de las velas rojas, nos pasamos de la restricción de no caer más del 10%.

Y como ven necesitamos 132 sucesos negativos para recién alcanzar la restricción de perder el 95%

Pero nuestra idea original no era calcular la cantidad de velas tolerada, sino la probabilidad de que esa cantidad de "velas rojas" seguidas aparezca. Lo de "velas rojas" entre comillas porque es una forma de decir en realidad puedo tener una vela verde con gap bajista y es un día perdedor, pero es para que se entienda mas conceptual lo de "vela roja" que es más visualizable que "suceso con resultado de payoff negativo"

Bueno, ya casi, solo falta una fórmula, la separé en dos para no armar tanto kilombo despejando con logaritmos porque si no, queda muy fea y no es tan así.

$$RoR = \left(\frac{1 - (W - L)}{1 + (W - L)} \right)^U$$

Donde:

RoR: Es la probabilidad de sobrepasar la restricción de "max_perdida" seteada para el cálculo de U

U: Es la cantidad máxima de "velas rojas" o trades perdedores soportada para no romper restricción mencionada

W: Es la probabilidad de suceso ganador

L: Es 1-W, o la probabilidad de suceso perdedor

Aplicando entonces las fórmulas vemos cual es el riesgo de ruina seteando ese 95% de perdida "ruina"

```
trade_perdedor = -strategy_lin[strategy_lin < 0].mean()  
  
max_perdida = 0.95  
U = int(np.log(1-max_perdida) / np.log(1-trade_perdedor))  
  
ganadores = strategy_lin[strategy_lin > 0]  
  
W = len(ganadores) / len(strategy_lin.loc[strategy_lin != 0])  
L = 1 - W  
  
RoR = ((1 - (W - L)) / (1 + (W - L))) **U  
f"Riesgo Ruina estrategia {RoR:.4%}"  
  
'Riesgo Ruina estrategia 0.0000%'  
  
trade_perdedor = -benchmark_lin[benchmark_lin < 0].mean()  
  
max_perdida = 0.95  
U = int(np.log(1-max_perdida) / np.log(1-trade_perdedor))  
  
ganadores = benchmark_lin[benchmark_lin > 0]  
  
W = len(ganadores) / len(benchmark_lin.loc[benchmark_lin != 0])  
L = 1 - W  
  
RoR = ((1 - (W - L)) / (1 + (W - L))) **U  
f"Riesgo Ruina bechmark {RoR:.4%}"  
  
'Riesgo Ruina bechmark 0.0063%'
```

Obviamente la probabilidad es muy baja, de hecho 0.000 varios ceros y algo. Pero claro, es muy difícil al azar tomando valores de TSLA que tengamos tanta mala suerte de agarrar 133 seguidos negativos, ya hablamos dicho que era virtualmente imposible con este U (y ese U sale no solo del 95% seteado sino también de las velas medias de TSLA los días que pierde). Esto se vuelve relevante y muy relevante en el trading de opciones o criptomonedas porque las variaciones son realmente mucho mayores, pero con variaciones donde el promedio de las negativas no supera el -2.2% no es un problema.

De todos modos, calculemos la probabilidad de perforar un valor más normal para tradear este tipo de activos, por decir algo tomemos el 10%

```
trade_perdedor = -strategy_lin[strategy_lin < 0].mean()

max_perdida = 0.1
U = int(np.log(1-max_perdida) / np.log(1-trade_perdedor))

ganadores = strategy_lin[strategy_lin > 0]

W = len(ganadores) / len(strategy_lin.loc[strategy_lin != 0])
L = 1 - W

RoR = ((1 - (W - L)) / (1 + (W - L))) **U
f"Riesgo Ruina estrategia {RoR:.4%}"

'Riesgo Ruina estrategia 49.7732'

trade_perdedor = -benchmark_lin[benchmark_lin < 0].mean()

max_perdida = 0.1
U = int(np.log(1-max_perdida) / np.log(1-trade_perdedor))

ganadores = benchmark_lin[benchmark_lin > 0]

W = len(ganadores) / len(benchmark_lin.loc[benchmark_lin != 0])
L = 1 - W

RoR = ((1 - (W - L)) / (1 + (W - L))) **U
f"Riesgo Ruina bechmark {RoR:.4%}"

'Riesgo Ruina bechmark 74.9160'
```

Y vean como cambia la cosa ahora, si calculamos la probabilidad de perder más del 10% en una simulación, en el caso de nuestra estrategia esa proba es 49.77% mientras que en el buy&Hold de TSLA es del 74.91%

A ver, no es la esperanza matemática de invertir en TSLA, tiene que quedar claro ese concepto, es simplemente la probabilidad de que tomando muchas simulaciones al azar aparezca una de todas esas simulaciones que rompa la barrera de pérdida máxima en este caso el 10%.

Obviamente que entre todas las simulaciones van a haber muchas que ganen mucho, el tema acá es evaluar el riesgo de perforar cierta barrera.

En el caso de TSLA y el 10% es un riesgo alto, pero vemos como la estrategia lo mitiga bastante bien, o sea como veníamos viendo en otros indicadores de riesgo, la estrategia disminuye el riesgo de exposición a TSLA respecto al buy&Hold.

Value at Risk VaR

Llegamos a la métrica de riesgo más utilizada en los libros financieros clásicos y los de risk management, el famoso "Vale at Risk", conceptualmente es una métrica que intenta mostrarnos a que estamos expuestos en términos nominales o porcentuales, es decir ponele un VaR=8% sería como decirte

"mirá que cada día estas arriesgando el 8% de tu portafolio eh"

Bueno, algo así, pero como veremos tiene miles de "peros" o cositas a tener en cuenta en función de las muchas formas de calcularlo.

Lo mismo que en el caso de los criterios de ruina, acá podemos asumir un determinado modelado de distribución o bien podemos tomar los valores empíricos

VaR con valores empíricos

En este caso es sencillo, simplemente calculamos un determinado quantil, por ejemplo uno muy usado es el 0.05 es decir calcular el valor de retorno para el cual hay menos de 5% de probabilidad de retorno menor al dado

Con pandas esto de super sencillo, simplemente hacemos:

```
var_emp = strategy_lin.quantile(.05)
var_emp
-0.034192767615613934

strategy_lin.quantile(.95)
0.040394901134818306
```

Esto nos indica que en el 90% de los casos, los retornos estarán entre -3.4192% y +4.0394%

Calculamos entonces el VaR con los valores empíricos tomando un intervalo de confianza del 90%, es decir, el quantil 0.05 tanto para la estrategia como para el benchmark:

```
ic = 0.9
var_emp_strategy = strategy_lin.quantile((1-ic)/2)
var_emp_benchamrk = benchmark_lin.quantile((1-ic)/2)

print(f'El VaR empírico con un Intervalo de confianza del {ic:.0%} es:')
print(f' -Para la estrategia es {var_emp_strategy:.2%}')
print(f' -Para el Buy&Hold es {var_emp_benchamrk:.2%}')
```

El VaR empírico con un Intervalo de confianza del 90% es:

-Para la estrategia es -3.42%
-Para el Buy&Hold es -4.72%

Seguimos con el mismo resultado que los otros indicadores de riesgo, super coherente por ahora, vemos que el VaR es menor en nuestra estrategia que en el benchmark que era el Buy&Hold de TSLA que es el subyacente tradeado. Tiene todo el sentido del mundo, pero bueno son diferentes indicadores de riesgo, la idea del VaR es tener un número de "dolor esperable" cualquier día al estar expuesto a la estrategia o benchmark, es decir que con un VaR de -3.42% estoy esperando "ese dolor" en un 5% de los días (porque configuré el intervalo de confianza en 90%), obviamente si soy mas estricto y configuro el IC en 99% para estar mas seguro de no pasar por "el dolor", el VaR será mayor (más negativo quiero decir, claro) pero ocurrirá menos seguido.

O sea, si estoy dispuesto a sufrir hasta -3.42% en el 5% de los días, esta estrategia tiene un riesgo tolerable para mí, esa es un poco la idea del VaR.

Calculemos como declara lo mismo para un IC del 99%

```
ic = 0.99
var_emp_strategy = strategy_lin.quantile((1-ic)/2)
var_emp_benchamrk = benchmark_lin.quantile((1-ic)/2)

print(f'El VaR empírico con un Intervalo de confianza del {ic:.0%} es:')
print(f' -Para la estrategia es {var_emp_strategy:.2%}')
print(f' -Para el Buy&Hold es {var_emp_benchamrk:.2%}')

El VaR empírico con un Intervalo de confianza del 99% es:
-Para la estrategia es -8.88%
-Para el Buy&Hold es -11.60%
```

O sea que en el 0.5% de las veces voy a tener que estar dispuesto a tolerar un -8.88% es decir un día cada 200. Mas o menos una vez al año, si no estoy dispuesto a tolerar un día esa caída claramente no debería aceptar ese nivel de riesgo, y mucho menos en el buy&Hold que es mayor aun.

Pero esto es todo con valores empíricos, lamentablemente el mainstream va medio contramano y usa modelado y para peor, modela con una distribución normal, personalmente creo que es una manera que adoptó la industria de vender algo menos riesgoso de lo que en realidad es sobre todo con intervalos de confianza altos como el 99%, pero bueno, no soy quien para discutirlo, solo doy mi opinión, así que veamos a continuación el cálculo del VaR asumiendo una distribución normal.

VaR asumiendo normalidad

En este caso lo que vamos a hacer es tomar los rendimientos (tanto para la estrategia como para el benchmark) y calcular con ellos su media y desvío estándar, y luego usaremos scipy para tomar como si fuera una normal con estos parámetros para luego sí, calcular el VaR, ya no con los datos empíricos, sino con esa distribución normal de ese mu y ese sigma.

Para ello usaremos la PPF que es la inversa de la CDF (lo mismo que hicimos con lo empírico, dado un valor y la proba de ocurrencia de ese valor) La PPF tiene en X los valores acumulados de probabilidad y en Y los valores de rendimientos para esos valores de X. En cambio, la CDF tiene en X los valores de rendimientos y en Y los acumulados de probabilidad para esos X.

Todo este tema lo pueden repasar del t[2] y t[6]

Pero se va a entender mejor con el código directamente así que vamos a ello:

```
from scipy.stats import norm

ic = 0.99

var_mod_strategy = norm.ppf((1-ic)/2, loc=strategy_lin.mean(), scale=strategy_lin.std())
var_mod_benchmark = norm.ppf((1-ic)/2, loc=benchmark_lin.mean(), scale=benchmark_lin.std())

print(f'El VaR Modelando Normal, con un Intervalo de confianza del {ic:.0%} es:')
print(f' -Para la estrategia es {var_mod_strategy:.2%}')
print(f' -Para el Buy&Hold es {var_mod_benchmark:.2%}')

El VaR Modelando Normal, con un Intervalo de confianza del 99% es:
-Para la estrategia es -4.02%
-Para el Buy&Hold es -5.49%
```

Como ven la librería quantstats, usa tal cual el mainstream el modelado asumiendo normalidad y un intervalo de confianza del 90% para el cálculo del VaR y nos da tal cual como lo calculamos en la página anterior

Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.99%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (cVaR)	-4.02%	-5.49%
Payoff Ratio	1.17	1.17
Profit Factor	1.4	1.24
Common Sense Ratio	1.65	1.44
CPC Index	0.89	0.76
Tail Ratio	1.18	1.15
Outlier Win Ratio	10.74	4.1
Outlier Loss Ratio	3.62	3.67

Pero también podríamos calcular el VaR para un Intervalo de confianza del 99%:

```
from scipy.stats import norm

ic = 0.99

var_mod_strategy = norm.ppf((1-ic)/2, loc=strategy_lin.mean(), scale=strategy_lin.std())
var_mod_benchmark = norm.ppf((1-ic)/2, loc=benchmark_lin.mean(), scale=benchmark_lin.std())

print(f'El VaR Modelando Normal, con un Intervalo de confianza del {ic:.0%} es:')
print(f' -Para la estrategia es {var_mod_strategy:.2%}')
print(f' -Para el Buy&Hold es {var_mod_benchmark:.2%}')

El VaR Modelando Normal, con un Intervalo de confianza del 99% es:
-Para la estrategia es -6.41%
-Para el Buy&Hold es -8.75%
```

Y vemos, como en lugar del -11.6% que daba el Buy&Hold con los datos empíricos, ahora con el 99% de confianza asumiendo normalidad baja a un -8.75% el VaR, como les decía asumir normalidad, camufla los riesgos de cola, con lo cual, mientras mas alto sea el intervalo de confianza mas "mentiroso" será el VaR calculado con esta suposición errónea de normalidad. Al ser un ratio que supuestamente mide el riesgo de cola me parece personalmente un despropósito asumir normalidad para su cálculo a sabiendas del punto débil que tiene justamente en estas colas, pero bueno, la mayoría de la bibliografía usa este formato por eso lo muestro pero no sin dejar de marcar mi opinión, perdón la insistencia, pero no puedo dejar de mencionarlo.

cVaR (valor en riesgo condicional) - Expected Shortfall

Esto es un poco mas difícil de explicar con palabras pero voy a intentarlo, el VaR mira digamos el punto que se encuentra justo en el quantil 0.05, es decir de ahí para abajo no importa "que tan abajo estén los valores" sino que solo mira ese valor justo posado en el $q=0.05$, esto es medio engañoso porque supongamos que ese valor de retorno sea como vimos en el Buy&Hold el -5.49%, esto significa que el 5% de las veces el retorno va a ser inferior a eso, pero no sabemos que tan inferior, no es lo mismo que ese 5% esté cerca por ejemplo en -6% a que la mayoría esté en -40%, claramente este segundo caso es mucho mas riesgoso e indeseable pero el VaR así como viene no nos dice nada de esa característica de "que tan lejos" pueden estar los retornos de ese VaR calculado.

Bueno, justamente esto viene a solucionar el cVaR también llamado "expected tail loss" ya que mide en definitiva de alguna manera la forma de esa cola de la distribución o que tan pesada es.

Y acá la misma historia de siempre, tenemos una forma de calcular esto asumiendo una distribución determinada (y lo mismo de siempre, una gran parte de la bibliografía asume normalidad, lo cual por lo general subestima porque asume que del valor dado para la izquierda de la distribución se va a comportar como una distribución normal que tiene colas mucho más "livianas que las distribuciones de retornos de acciones en la bolsa)

O bien podríamos calcular esto de una manera empírica así con los datos que tenemos sin asumir ninguna forma en particular, ya veremos cómo es el tema.

Pero arranquemos con lo mas sencillo que es asumir una distribución normal

Esta es la manera de calcularlo:

$$ES_{\alpha}(L) = \mu + \sigma \frac{\varphi(\Phi^{-1}(\alpha))}{\alpha}$$

Donde:

- μ es la media de los retornos
- σ es el desvío estándar de los retornos
- φ es la PDF (función de densidad de probabilidades)
- Φ es la CDF (Función de probabilidades acumulada)
- α es el valor del quantil a calcular (de ahí para la izquierda obviamente)

Bueno, mucho chirimbolo con letras griegas pero vamos a calcular esto en Python que se ve más claro así:

```
mu = strategy_lin.mean()
sigma = strategy_lin.std()

alpha = 0.05
cVar_mod_norm = -mu + sigma * norm.pdf(norm.cdf(alpha)**-1)/(alpha)
print(f"cVaR Modelado Normal para la estrategia: {-cVar_mod_norm:.2%}")

cVaR Modelado Normal para la estrategia: -3.02%

mu = benchmark_lin.mean()
sigma = benchmark_lin.std()

alpha = 0.05
cVar_mod_norm = -mu + sigma * norm.pdf(norm.cdf(alpha)**-1)/(alpha)
print(f"cVaR Modelado Normal para el benchmark: {-cVar_mod_norm:.2%}")

cVaR Modelado Normal para el benchmark: -4.13%
```

Como ven nuevamente obtenemos la misma data que antes con el VaR simple, que es el hecho que la estrategia es menos arriesgada que el Buy&Hold, pero bueno, como dijimos estamos asumiendo normalidad cosa que no está bueno

Y ya que estamos vamos a calcular esto con un quantil mas exigente por ejemplo en el $q=0.01$, es decir con un 98% de confianza, para ello voy a mejorar un poquito el código poniendo el intervalo de confianza como una variable a definir inicialmente, como hice en los ejemplos anteriores, no quise de una complicar mucho las fórmula pero ahí va mejorado.

Y ya que estamos lo metemos en una función

```
def cVaR_norm(returns, IC=0.9):
    mu = returns.mean()
    sigma = returns.std()
    alpha = (1-IC)/2

    cvar_mod_norm = -mu + sigma * norm.pdf(norm.cdf(alpha)**-1)/(alpha)
    print(f"cVaR Modelado Normal: {-cvar_mod_norm:.2%}, int confianza={IC:.0%}")
    return -cvar_mod_norm

print('cVaR asumiendo normalidad para Estrategia y Benchmark respectivamente')
cVaR_norm(strategy_lin, IC=0.98)
cVaR_norm(benchmark_lin, IC=0.98)

cVaR asumiendo normalidad para Estrategia y Benchmark respectivamente
cVaR Modelado Normal: -14.11%, int confianza=98%
cVaR Modelado Normal: -19.22%, int confianza=98%
```

Como ven pasando como argumento un intervalo de confianza del 98% para obtener el cVaR con el quantil=0.01 nos da valores bastante más aberrantes.

La lectura de almacenero de esto sería que "en el 1% de las velas el promedio de pérdida sería del 14.11% para la estrategia o del 19.22% en promedio ese 1% de velas para el Buy&Hold"

El problemita de esto es que asumimos que del lado izquierdo de ese quantil 0.01 se comporta como una normal y calculamos ese "promedio" con esa suposición, veamos entonces la otra forma de calcular esto sin asumir normalidad

cVaR, cálculo empírico

Primero entendamos el concepto, si esto de los retornos y sus VaR fuera una función continua simplemente tendríamos una expresión como la siguiente, el tema es que ni es una función continua, ni la tenemos, ni queremos asumir un tipo de distribución que sea solo teórica, ya que dijimos que vamos a calcular el cVaR de los datos empíricos sin asumir nada, así que deberemos buscar una forma más realista

$$ES_{\alpha}(X) = -\frac{1}{\alpha} \int_0^{\alpha} \text{VaR}_{\gamma}(X) d\gamma$$

Bajando esto a una expresión discreta con los valores dados, lo voy a explicar a lo almacenero para que se entienda, lo que tenemos que hacer es agarrar todos los valores a la izquierda de nuestro " α " y promediar sus retornos, tan sencillo como eso, veamos una implementación primero bien burda para entenderla:

Ahora ¿cuáles son los valores a la izquierda de ese " α "? asumiendo como $\alpha=0.05$ que queremos un intervalo de confianza del 90% es decir que queremos el cVaR de los valores menores al quantil=0.05

Esto es muy sencillo de hacer con pandas porque directamente filtramos la cola izquierda usando la función quantile(α) y luego simplemente promediamos los retornos de esa cola izquierda. Si nos ponemos finos en realidad no hay que hacer un promedio de los retornos lineales sino un expected value de los logarítmicos, pero en este tipo de valores sinceramente no hay mucha diferencia y de nuevo, como digo siempre, lo importante es usar el mismo criterio para comparar, no nos importa tanto el valor nominal sino la referencia comparativa.

Bueno veamos entonces como nos da el verdadero cVaR sin asumir normalidad a la izquierda de la cola

```
mu_strategy = strategy_lin.mean()
sigma_strategy = strategy_lin.std()

alpha = 0.05
pto_ref = strategy_lin.quantile(alpha)
tail_strategy = strategy_lin.loc[strategy_lin < pto_ref].tolist()
cVar_strategy = sum(tail_strategy) / len(tail_strategy)
cVar_strategy

-0.05805222200691991

mu_benchmark = benchmark_lin.mean()
sigma_benchmark = benchmark_lin.std()

alpha = 0.05
pto_ref = benchmark_lin.quantile(alpha)
tail_benchmark = benchmark_lin.loc[benchmark_lin < pto_ref].tolist()

cVar_benchmark = sum(tail_benchmark) / len(tail_benchmark)
cVar_benchmark

-0.07524536819301726
```

Bien, como vemos el cVaR sin asumir normalidad con el alpha=0.05 me da

cVaR estrategia = -5.85%

cVaR benchmark = -7.52%

Mientras que asumiendo normalidad me habia dado:

cVaR estrategia = -3.02%

cVaR benchmark = -4.13%

¿ven por qué me quejo tanto de esa suposición de normalidad en ratios de riesgo?

Claramente subestima demasiado y es lógico, ya que si asumo normalidad, abajo del q=0.05 voy a estar asumiendo una cola delgada o liviana que no es real, sino que la real es bastante mas pesada por lo general en activos financieros.

Vamos a calcular el cVaR con el q=0.01 y de paso, ahora que entendimos el concepto, vamos a pulir nuestro cálculo en una función mas pythónica

```
def cVaR(returns, IC=0.9):
    mu_benchmark = returns.mean()
    sigma_benchmark = returns.std()

    alpha = (1-IC)/2
    pto_ref = returns.quantile(alpha)
    tail = returns.loc[returns < pto_ref].tolist()

    cVaR = sum(tail) / len(tail)
    print(f'El cVaR empírico para un IC={IC:.0%} es:{cVaR:.2%}')
    return cVaR

print('cVaR empírico para Estrategia y Benchmark respectivamente')
cVaR(strategy_lin, IC=0.98)
cVaR(benchmark_lin, IC=0.98)

cVaR empírico para Estrategia y Benchmark respectivamente
El cVaR empírico para un IC=98% es:-10.08%
El cVaR empírico para un IC=98% es:-12.66%
```

PayOff Ratio

Este es quizás el ratio más representativo y repetido en los libros que tocan este tipo de temas, ya que mide en cierta forma la asimetría de los trades, lo que se intenta saber es si se gana más (cuando se gana) de lo que se pierde (cuando se pierde)

Es decir, es el cociente entre la media de los trades ganadores sobre la media de los perdedores.

Como casi siempre, voy a hacer un disclaimer que considero relevante, el tema es que en casi todos los lugares donde leo este tema del payoff ratio, nadie aclara que deben usarse los retornos logarítmicos, es más todos usan los retornos lineales, y está mal, no, está muy mal, bueno, que se yo, no está tan mal, bueno, ya era hora de otro meme jaja



El tema es que un trade de -50%, como ya sabemos, equivale simétricamente a uno de +100% en el otro lado, o sea, no puedo promediar los negativos por un lado y los positivos por el otro así felizmente, como si fuera simétrico todo... Ahora el tema es que, si solo lo uso para comparar la estrategia contra el benchmark, bueno, ahí a nivel comparativo no jode tanto que consideremos simétrico algo que no lo es porque en ambos casos estamos tomando el mismo sesgo. Ahora no me vengan con que el método es bueno porque el payoff ratio es >1 , diciendo que las ganadoras ganan más de lo que pierden las perdedoras porque insisto si en promedio las ganadoras (retornos lineales) dan +30% y las perdedoras en promedio -25%, el ratio les va a dar 1.2 lo cual parece muy bueno, pero en la práctica salen perdiendo porque $USD\ 100 + 30\% - 25\% = USD\ 97.5$

Ahora, el tema es que con rendimientos diarios, cuyas medias positivas y negativas rondan el 2% por decir algo, la verdad la diferencia entre retornos lineales y logarítmicos es medio despreciable, por eso digo que no está tan mal usar los lineales, además si solo lo usamos en términos comparativos, como decía, más irrelevante aún que tipo de retorno usamos ya que la gracia es comparar una estrategia contra un benchmark, o como veremos más adelante, con la misma estrategia parametrizada diferente.

Vamos directo al código ya que no hace falta fórmula acá por ser demasiado sencillo:

```
def payoffRatio(returns):
    perdedores = returns[returns < 0]
    ganadores = returns[returns > 0]
    payoff_ratio = -ganadores.mean() / perdedores.mean()
    return payoff_ratio

payoffRatio(strategy_lin), payoffRatio(benchmark_lin)
(1.1704400650339681, 1.15589427505077)

payoffRatio(strategy_log), payoffRatio(benchmark_log)
(1.111321313369147, 1.0980745345630865)
```

Como ven, hay una diferencia entre usar retornos lineales y logarítmicos, naturalmente los ratios con retornos lineales siempre dan mejor, esto es porque la asimetría es infinita para el lado positivo es decir puede haber una ganancia de 1000% o 5000% o más, pero para el lado negativo la máxima pérdida es -100%, con lo cual usar retornos lineales "ayuda" al ratio a estar arriba de 1 engañosamente, es decir si uso ratios lineales y me da arriba de 1, no es para ponerse contentos, si en cambio si se usan logarítmicos, ya que al ser simétricos los retornos logarítmicos, un payoffRatio >1 significaría que tendrímos una esperanza matemática positiva con este tipo de retornos.

Como les decía desde le principio lo importante es la comparación benchmark contra estrategia, o como veremos mas adelante en la parametrización, comparar distintas parametrizaciones de la misma estrategia, con lo cual si usamos lineales o logarítmicos pasa medio a un segundo plano porque lo que mas nos va a interesar es la comparativa, y si miran bien, en esa comparativa nos está dando 1.17 vs 1.15 (lineales) y 1.11 vs 1.09 (logarítmicos) es decir en ambos casos me da mejor el ratio en la estrategia vs el benchmark.

¿Qué significa que de mejor? Bueno conceptualmente que "en promedio" los días positivos gano mas de lo que pierdo los días negativos, pero ojo que ese "promedio" también puede ser engañoso, ¿por qué? Porque simplemente el promedio puede dar alto por pocos valores positivos y del lado negativo no es tan grave pero tenemos 20 veces mayor cantidad (mas allá del tema promedio/mediana y distribución, hay un tema de cantidades)

¿Para qué es útil este payoffRatio?

Fácil, para setear un buen ajuste de stop Loss o de take profit, dado que conocemos la media de cada lado, esto junto a datos de volatilidad y algunas cositas mas nos van a ser de gran ayuda para setear inicialmente este tipo de salidas, aunque obviamente un ajuste mas fino requiere de mucho mayor análisis que de simples ratios.

Como pueden observar, la librería usa los retornos lineales, y tiene un bug y me repite el valor de la estrategia en el benchmark, cuando dan diferente, como vimos en la estrategia da 1.17 pero en el benchmark da 1.155

Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.99%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (cVaR)	-4.02%	-5.49%

Payoff Ratio	1.17	1.17
Profit Factor	1.4	1.24
Common Sense Ratio	1.65	1.44
CPC Index	0.89	0.76
Tail Ratio	1.18	1.15
Outlier Win Ratio	10.74	4.1
Outlier Loss Ratio	3.62	3.67

Profit Factor

Habíamos visto que el payoff Ratio, tenía el problema que no media cantidades positivas vs negativas, sino simplemente el promedio de cada lado, pero si tengo un promedio positivo mucho mas alto que el negativo, esto me va a dar un excelente payoffRatio pero probablemente la cantidad negativa sea mucho mayor a la cantidad positiva y haga que por mas que pierda mucho menos que lo que gano en las buenas, el balance sea negativo.

Entonces viene este famoso Profit Factor, es un indicador muy útil en mi opinión, pero generalmente mal usado o mal interpretado, veámoslo con nuestro ejemplo y como lo calcula la librería quantstats.

La función que implementa es algo así:

```
def profitFactor(returns):
    perdedores = returns[returns < 0]
    ganadores = returns[returns > 0]
    profit_factor = -ganadores.sum() / perdedores.sum()
    return round(profit_factor,3)
```

Pero el problema es que usa como argumentos los retornos lineales:

```
profitFactor(strategy_lin), profitFactor(benchmark_lin)
(1.396, 1.245)
```

Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.96%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (CVaR)	-4.02%	-5.49%
Payoff Ratio	1.17	1.17
Profit Factor	1.4	1.24
Common Sense Ratio	1.65	1.44
CPC Index	0.89	0.76
Tail Ratio	1.18	1.15
Outlier Win Ratio	10.74	4.1
Outlier Loss Ratio	3.62	3.67

Sin embargo, esta haciendo un cociente de positivos y negativos asimétricos como los retornos lineales, con lo cual el ratio obtenido es engañoso, de nuevo, mientras sepamos y seamos conscientes de como lo estamos calculando está ok, pero prefiero personalmente calcularlo con logarítmicos

```
profitFactor(strategy_log), profitFactor(benchmark_log)
(1.325, 1.182)
```

Como ven disminuyen un poco los valores por lo que ya sabíamos, pero sigue dando mejor la estrategia que el benchmark, es decir que en ambos se compensan de mas las pérdidas con las ganancias, pero más acentuado en la estrategia la relación que en el buy&Hold.

Rachev Ratio

Llegó el turno de los ratios de cola, estos son particularmente útiles para medir el riesgo de esos eventos que mas nos preocupan a los inversores, que son los momentos críticos, es decir, quizás uno no se preocupa mucho de los días que baja nuestro portafolio un 0.algo % pero si le tenemos mucha aversión a esos días que todo se desploma y ahí es donde queremos estar más tranquilos, así que nada mejor para entender este tipo de riesgos que los ratios que miden riesgo de cola (tail risks)

Acá como veremos se abre toda otra discusión que es como medimos las colas, y como siempre arranca desde lo simplista que es más directo de calcular e interpretar pero quizás menos representativo, hasta lo más representativo de la realidad pero más abstracto en el cálculo, por lo que tendremos:

- Tail ratio por quantiles
- Tail ratio por medias
- Tail ratio por integrales

Veamos una por una estas formas de calcular un ratio de colas

Tail Ratio - Por quantiles

Se trata de simplemente medir quantiles simétricos a ambos extremos y hacer el cociente de los valores de la distribución representativos de esos quantiles, sin tener en cuenta el signo

Veamos la implementación en python:

```
def rachev_1(returns, alpha=0.05):
    tail_right = returns.quantile(1-alpha)
    tail_left = abs(returns.quantile(alpha))
    return round(tail_right / tail_left, 3)
```

Como ven solo tomamos el valor de un quantil digamos el 0.05 y el 0.95 por default y calculamos el ratio de donde se ubica cada uno de esos puntos en la distribución

Esto nos arroja este resultado:

```
print('Los Rachev Ratio para la estrategia y Benchmark son respectivamente:')
rachev_1(strategy_lin), rachev_1(benchmark_lin)

Los Rachev Ratio para la estrategia y Benchmark son respectivamente:
(1.181, 1.155)
```

Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.99%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (cVaR)	-4.02%	-5.49%
Payoff Ratio	1.17	1.17
Profit Factor	1.4	1.24
Common Sense Ratio	1.65	1.44
CPC Index	0.89	0.76
Tail Ratio	1.18	1.15
Outlier Win Ratio	10.74	4.1
Outlier Loss Ratio	3.62	3.67

Como pueden ver es lo que mas se usa, de hecho, es lo que usa la librería quanstats

Obviamente, misma observación de siempre, perdón, pero no puedo dejar de marcarlo, más en temas de colas, sería conveniente usar retornos logarítmicos:

```
print('Los Rachev Ratio para la estrategia y Benchmark son respectivamente:')
print('Esta vez usando retornos logarítmicos')
rachev_1(strategy_log), rachev_1(benchmark_log)

Los Rachev Ratio para la estrategia y Benchmark son respectivamente:
Esta vez usando retornos logarítmicos
(1.138, 1.098)
```

Pero rápidamente, más allá del tema retornos lineales vs logarítmicos, notarán que el valor que aparece en "x" en el quantil 0.05 no me dice nada porque a la izquierda de ese quantil puedo tener valores todos muy cercanos a ese 0.05 lo cual no sería tan grave o tener valores 10 sigmas hacia la izquierda que harían una cola mucho más pesada y peligrosa

Esto nos lleva a tener que buscar la forma de meter en el cálculo todo lo que hay del otro lado de ese quantil y nos lleva a los siguientes ratios de riesgo de cola

Tail Ratio – Por medias

Como imagináran lo que podríamos hacer es tomar todo lo que está a la izquierda del quantil 0.05 y promedia todos esos valores como si se tratara de una distribución aparte, y lo mismo con el quantil 0.95 y todo lo que esté a su derecha, y luego hacer el ratio entre ambos

Bueno, no es ni mas ni menos que eso:

```
def rachev_2(returns, alpha=0.05):
    tail_right = returns[returns > returns.quantile(1-alpha)]
    tail_left = returns[returns < returns.quantile(alpha)]

    return round(tail_right.mean() / tail_left.abs().mean(), 3)

print('Los Rachev Ratio para la estrategia y Benchmark son respectivamente:')
rachev_2(strategy_lin), rachev_2(benchmark_lin)
Los Rachev Ratio para la estrategia y Benchmark son respectivamente:
(1.247, 1.21)

print('Los Rachev Ratio para la estrategia y Benchmark son respectivamente:')
print('Esta vez usando retornos logarítmicos')

rachev_2(strategy_log), rachev_2(benchmark_log)
Los Rachev Ratio para la estrategia y Benchmark son respectivamente:
Esta vez usando retornos logarítmicos
(1.151, 1.098)
```

Esto ya es bastante mas representativo que solo el quantil 0.05 vs el 0.95

Pero si nos ponemos un poco mas finos, tampoco es realista porque ese promedio (por más que use retornos logarítmicos) puede ser medio mentiroso porque puede estar muy influenciado por un solo outlier muy sacado de contexto, o peor aún, un spike o un error del feed de datos. Esto se puede solucionar con un previo filtro de los datos o bien con un cálculo que contemple de aluna manera valor por valor y no una paramétrica como la media o mediana que solucionaría el tema outliers pero no el tema de la forma de la distribución.

Tail Ratio – Por integrales

Entonces, como decíamos, si nos ponemos mas finos, lo que vamos a querer es ver valor por valor a cada lado de la cola, cuanto vale cada "x" pero también que tan probable es su ocurrencia en una distribución normal (o el modelado que asumimos y de nuevo acá repasar el t[6] y ver toda la tela que hay para cortar al respecto)

La implementación básica asumiendo normalidad sería algo así:

```
from scipy.stats import norm

def rachev_3(returns, alpha=0.05):
    q_inf, q_sup = returns.quantile(0.05), returns.quantile(0.95)
    left = [abs(x) for x in returns if x < q_inf]
    right = [x for x in returns if x > q_sup]
    return (norm.pdf(right) * right).sum() / (norm.pdf(left) * left).sum()
```

Y si calculamos tanto para retornos lineales como logarítmicos obtenemos:

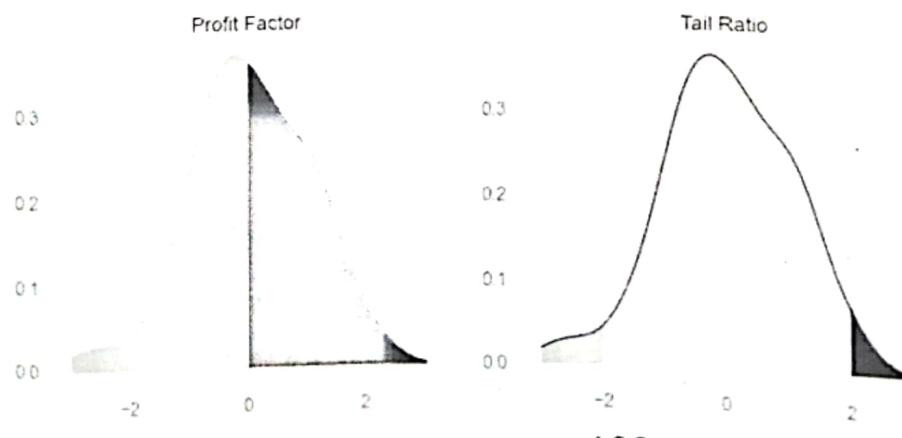
```
print('Los Rachev Ratio para la estrategia y Benchmark son respectivamente:')  
rachev_3(strategy_lin), rachev_3(benchmark_lin)  
Los Rachev Ratio para la estrategia y Benchmark son respectivamente:  
(1.2447167521329772, 1.207596355044497)  
  
print('Los Rachev Ratio para la estrategia y Benchmark son respectivamente:')  
print('Esta vez usando retornos logarítmicos')  
rachev_3(strategy_log), rachev_3(benchmark_log)  
Los Rachev Ratio para la estrategia y Benchmark son respectivamente:  
Esta vez usando retornos logarítmicos  
(1.1503164040010185, 1.0978474042402036)
```

O sea que finalmente, esos valores 1.1503 vs 1.0978 estrategia vs benchmark son los mas realistas y personalmente los que mas información creo que me dan.

De nuevo, me dirán toda esta vuelta para eso usaba el 1.18 vs 1.15 del ratio mas burdo con los quantiles directos y listo, bueno, para eso está todo este set de herramientas, para razonarlas y usar a conveniencia lo que crean más útil en cada caso.

Vamos a cerrar el tema con una interpretación gráfica con un ejemplo trivial aleatorio para entender visualmente los ratios que acabamos de calcular:

```
from scipy.stats import norm, gaussian_kde  
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots(figsize=(10,4), ncols=2)  
  
# Armo distribucion aleatoria  
dist = pd.Series(norm(0,1).rvs(10**2))  
  
# Genero un kernel de funcion de densidad  
kde = gaussian_kde(dist)  
  
# Genero puntos del eje x equidistantes  
x = np.linspace(-3, 3, 1000)  
  
ax[0].fill_between(x, kde(x), where=x>0, color='k')  
ax[0].fill_between(x, kde(x), where=x<0, color='silver')  
ax[0].set_title('Profit Factor')  
  
ax[1].plot(x, kde(x), '-k', lw=1)  
ax[1].fill_between(x, kde(x), where=x>2, color='k')  
ax[1].fill_between(x, kde(x), where=x<-2, color='silver')  
ax[1].set_title('Tail Ratio')
```



En ambos casos la forma correcta de calcular los ratios sería dividiendo las áreas negras por las grises

Y ya que estamos que vimos esto conceptualmente en un ejemplo trivial veamos como es el gráfico con los zscores de las distribuciones reales de la estrategia versus el buy&Hold:

```

fig, ax = plt.subplots(figsize=(15,7), ncols=2, nrows=2)

zscores_strategy = (strategy_lin - strategy_lin.mean()).divide(strategy_lin.std()).dropna()
zscores_benchmark = (benchmark_lin - benchmark_lin.mean()).divide(benchmark_lin.std()).dropna()

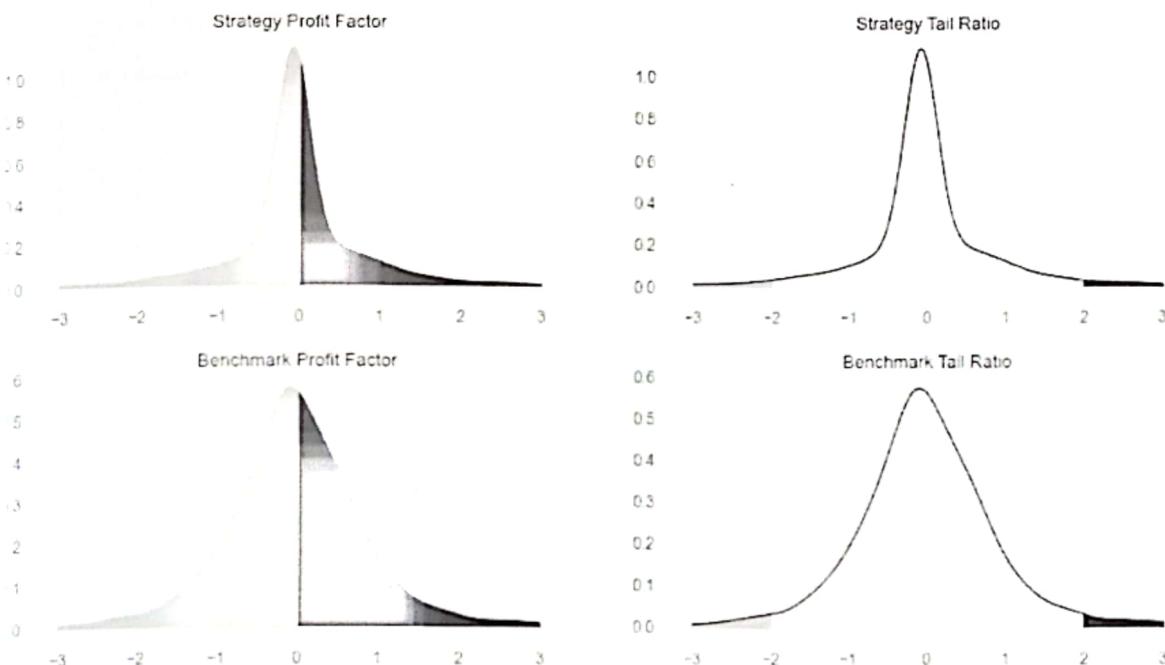
x = np.linspace(-3, 3, 1000)
kde = {}
kde['Strategy'] = gaussian_kde(pd.Series(zscores_strategy))
kde['Benchmark'] = gaussian_kde(pd.Series(zscores_benchmark))
keys = list(kde.keys())

for row in range(2):
    ax[row][0].fill_between(x, kde[keys[row]](x), where=x>0, color='k')
    ax[row][0].fill_between(x, kde[keys[row]](x), where=x<0, color='silver')
    ax[row][0].set_title(f'{keys[row]} Profit Factor')

    ax[row][1].plot(x, kde[keys[row]](x), '-k', lw=1)
    ax[row][1].fill_between(x, kde[keys[row]](x), where=x>2, color='k')
    ax[row][1].fill_between(x, kde[keys[row]](x), where=x<-2, color='silver')
    ax[row][1].set_title(f'{keys[row]} Tail Ratio')

plt.subplots_adjust(hspace=0.3)

```



Creo que se entiende mejor con los gráficos, y esto me lleva a la ultima vuelta de tuerca de estos ratios que combina a ambos y es el producto del Profit Factor * Tail Ratio

Common Sense Ratio

Como anticipamos es el producto CSR = Profit Factor * Tail Ratio

La idea es resumir en un solo ratio que tanto se gana comparado con lo que se pierde, a su vez potenciado por el mismo factor en los extremos, veamos la implementación de la forma mas sencilla en python

Como ya teníamos funciones para el profitFactor y para los ratios de riesgo de cola los usamos:

```
def CSR(returns):
    return round(profitFactor(returns) * rachev_1(returns), 2)

CSR(strategy_lin), CSR(benchmark_lin)
(1.65, 1.44)
```

Esta es la forma mas directa y de hecho es la que usa la mayoría y la que usa quantstats, se los remarco para que chequen

Expected Daily %	0.18%	0.2%
Expected Monthly %	3.73%	4.2%
Expected Yearly %	52.99%	61.11%
Kelly Criterion	15.43%	10.71%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-4.02%	-5.49%
Expected Shortfall (cVaR)	-4.02%	-5.49%
Payoff Ratio	1.17	1.17
Profit Factor	1.4	1.24
Common Sense Ratio	1.65	1.44
CPC Index	0.89	0.76
Tail Ratio	1.18	1.15
Outlier Win Ratio	10.74	4.1
Outlier Loss Ratio	3.62	3.67

Pero ya me conocen yo hubiera hecho algo así:

```
def CSR(returns):
    return round(profitFactor(returns) * rachev_3(returns), 2)

CSR(strategy_log), CSR(benchmark_log)
(1.52, 1.3)
```

Mas ratios de riesgo

Por supuesto que esto no es todo, hay muchos ratios mas pero digamos que hasta acá vimos bastantes y quiero dedicarle mas espacio a las siguientes etapas del backtesting, simplemente nombro algún ratio mas interesante para que investiguen por su cuenta si quieren profundizar:

- Alpha Jensen
- Outliers Ratios
- Leverages Ratios
- Capital efficiency ratios
- Treynor Ratio

Y todo esto sin nombrar nada de métricas "de performance" y overfitting de modelos, que cuando toquemos los temas de entrenamiento de modelos, inteligencia artificial, machine learning y demás veremos. Acá solo estamos hablando de métricas de riesgo intrínseco al método de inversión, no al modelo de backtesting

Tablero de control, o Dashboard

Finalmente armaremos un tablero final del método backtesteado donde veremos las métricas más relevantes del modelo, gráficas etc, es como una salida final donde vemos lo más relevante, generalmente se suele poner gráficas de payoff comparando Estrategia vs BenchMark, una gráfica comparando año a año la performance de ambos, algo de estacionalidad mensual suele ser útil también y algo de retorno ajustado por riesgo que puede ser ajustado por volatilidad o alguna otra métrica de las que vimos, es super personal esta parte, solo les nombro cosas típicas, y por supuesto tablas de trades y retornos de cada operación.

Una cosa que a mi personalmente me gusta tener en el dashboard es un gráfico que muestre las entradas y salidas al menos de una ventana reciente de tiempo para poder observar a grandes rasgos si el método hace algo parecido a lo que nosotros pensábamos que iba a hacer en cuanto a entradas y salidas, porque muchas veces pasa que los números son muy fríos y no nos damos cuenta que lo que termina haciendo o ejecutando el método es algo que absolutamente nada que ver con lo que nos imaginábamos y este tipo de gráficos suele ser muy útil para detectar estos errores.

O sea, teníamos esto, valiéndonos de las funciones que ya vimos antes:

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                     trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

trades = getTrades(actions)
resumen(trades)
```

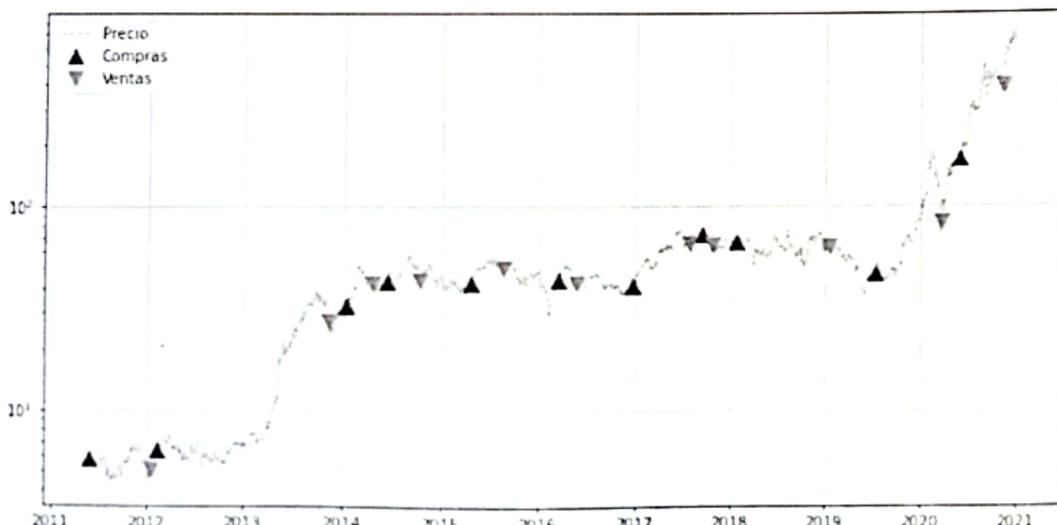
resultado	Ganador	Perdedor
Cantidad	7.000000	4.000000
Rendimiento x Trade	0.922483	-0.074515
Días Total	1606.000000	692.000000
Días x Trade	229.428571	173.000000
{'rendimiento': 29.8979, 'dias_in': 2298, 'TEA': 0.7245)}		

Y agregamos además de todas las métricas que vimos en detalle, alguna gráfica de este estilo:

```
df = data.iloc[:,:]
compras = actions.loc[actions.gatillo=='compra']
ventas = actions.loc[actions.gatillo=='venta']

fig, ax = plt.subplots(figsize=(12,6))

ax.plot(df.Close, '--k', alpha=0.4, lw=1, label='Precio')
ax.plot(compras.Close * 0.95, marker='^', lw=0, markersize=10, color='k', label='Compras')
ax.plot(ventas.Close * 0.95, marker='v', lw=0, markersize=10, color='red', label='Ventas')
ax.legend()
ax.set_yscale('log')
ax.grid(axis='both')
```



Etapa de parametrización

Bueno, llegamos a la siguiente etapa de un backtesting que es, una vez definido el racional, los features, y las métricas a utilizar para elegir y parametrizar features, comenzar a afinar variables.

Ejemplo, teníamos algo así:

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow=60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                      trig_sell_cross = -0.01, trig_sell_rsi=55) trig_sell_obv = 0

trades = getTrades(actions)
resumen(trades)
```

resultado	Ganador	Perdedor
Cantidad	7.000000	4.000000
Rendimiento x Trade	0.922483	-0.074515
Días Total	1606.000000	692.000000
Días x Trade	229.428571	173.000000
'rendimiento': 29.8979, 'dias_in': 2298, 'TEA': 0.7245}		

Bueno, todo eso que marco con círculos son en realidad valores a parametrizar, es decir, valores que tengo que estudiar que tan sensible es mi método al cambio de los mismos, y como en tantas cosas de este campo acá no hay una regla a seguir que siempre funciones o que asegure los mejores resultados, sino que dependerá de mil cosas, incluso haciendo el esfuerzo a adaptarse mentalmente para un ejercicio mas realista, ya que los ejemplos que vuelco en el libro, por una cuestión pedagógica son bastante triviales, es decir que en la realidad uno hace cosas bastante mas completas pero la idea en el libro es ver conceptos, luego las ideas de trading y los features, como vimos en todo un apartado en este mismo libro, deben ser mucho mas amplias que un cruce un RSI y no se que indicador de volumen, pero bueno, no me extiendo más con esta aclaración solo lo dejo como advertencia de que armen cosas más interesantes, bueno, sigamos adelante

Entonces, la primera pregunta es ¿por dónde empezamos a parametrizar todo eso?

Bueno, hay algos de machine learning que ayudan bastante, pero como aún no llegamos, vamos a ir viendo la forma manual de ir estudiando esto, pero les dejo al menos un esquema de pasos a seguir para que quede para más adelante, o bien para los que estén más avanzados en la parte de machine learning tengan un puntapié inicial para empezar a progresar a partir de estos bullet:

- Introspección inicial parametrizando feature por feature aparte
- Montecarlo parametrizando varios features a la vez
- Algoritmos genéticos, adaptativos y progresivos para optimización.
- Algos de machine learning del estilo PCA para reducir dimensionalidad
- Algos de ML del estilo arboles de decisión para parametrizar ayudados en métricas de performance de accuracy etc.
- Parametrización, Hiperparámetros y Cross Validation

Los dos primeros bullet los vamos a ver en este tomo, obviamente el resto queda para el tomo de IA bastante mas adelante luego de pasar previamente por el tomo de bases de datos e infraestructura por ejemplo.

previo a comenzar vamos a repasar las funciones que vamos a necesitar antes de parametrizar, se las pego acá para que no tengan que buscarla como 50 páginas atrás:

Primero obtener data y features a utilizar en nuestro método de swing trading:

```
def getData(ticker, data_from, data_to):
    data = yf.download(ticker, auto_adjust=True, progress=False,
                       start=data_from, end=data_to)
    return data

def getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60):
    data['Balance'] = np.where(data.Close>data.Close.shift(), data['volume'],
                               np.where(data.Close < data.Close.shift(), -data['volume'], 0))

    data['OBV'] = data['Balance'].cumsum()
    dif = data['Close'].diff()
    win = pd.DataFrame(np.where(dif > 0, dif, 0), index=data.index)
    loss = pd.DataFrame(np.where(dif < 0, abs(dif), 0), index=data.index)
    ema_win = win.ewm(alpha=1/n_rsi).mean()
    ema_loss = loss.ewm(alpha=1/n_rsi).mean()
    rs = ema_win / ema_loss

    data['cruce'] = data.Close.rolling(fast).mean() / data.Close.rolling(slow).mean() -1
    data['rsi'] = 100 - (100 / (1+rs))
    data['sigma'] = data.Close.pct_change().rolling(n_sigma).std()
    data['OBV_osc'] = (data.OBV - data.OBV.rolling(n_obv).mean()) / (data.OBV.rolling(n_obv).std())
    features = data.loc[:,['cruce','rsi','sigma','OBV_osc']].dropna()

    return features
```

Luego los triggers o acciones de compra/venta

```
def getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
               trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0):

    gatillos_compra = pd.DataFrame(index = features.index)
    gatillos_compra['cruce'] = np.where(features.cruce > trig_buy_cross, True, False)
    gatillos_compra['rsi'] = np.where(features.rsi > trig_buy_rsi, True, False)
    gatillos_compra['sigma'] = np.where(features.sigma > trig_buy_sigma, True, False)
    mascara_compra = gatillos_compra.all(axis=1)

    gatillos_venta = pd.DataFrame(index = features.index)
    gatillos_venta['cruce'] = np.where(features.cruce < trig_sell_cross, True, False)
    gatillos_venta['rsi'] = np.where(features.rsi < trig_sell_rsi, True, False)
    gatillos_venta['obv'] = np.where(features.OBV_osc > trig_sell_obv, True, False)
    mascara_venta = gatillos_venta.all(axis=1)

    data_aux = data.copy().dropna()
    data_aux['gatillo'] = np.where(mascara_compra, 'compra', np.where(mascara_venta, 'venta', ''))
```

actions = data_aux.loc[data_aux.gatillo != ''].copy()

```
actions['gatillo'] = np.where(actions.gatillo != actions.gatillo.shift(), actions.gatillo,'')
actions = actions.loc[actions.gatillo != ''].copy()
```

```
if actions.iloc[0].loc['gatillo'] == 'venta':
    actions = actions.iloc[1:]

if actions.iloc[-1].loc['gatillo'] == 'compra':
    actions = actions.iloc[:-1]
```

```
return actions
```

Finalmente una tabla de trades finales para poder llegar a una tabla resumen luego

```
def getTrades(actions):
    pares = actions.iloc[:,2].loc[:,['Close']].reset_index()
    impares = actions.iloc[1::2].loc[:,['Close']].reset_index()
    trades = pd.concat([pares, impares], axis=1)

    CT = 0
    trades.columns = ['fecha_compra', 'px_compra', 'fecha_venta', 'px_venta']
    trades['rendimiento'] = trades.px_venta / trades.px_compra - 1
    trades['rendimiento'] -= CT
    trades['dias'] = (trades.fecha_venta - trades.fecha_compra).dt.days

    if len(trades):
        trades['resultado'] = np.where(trades['rendimiento'] > 0, 'Ganador', 'Perdedor')
        trades['rendimientoAcumulado'] = (trades['rendimiento']+1).cumprod()-1

    return trades
```

No vamos a usar event-driven para parametrizar porque requiere mayor poder de cómputo ya que necesitamos lógica iterativa, esto se suele hacer con otros lenguajes que son mas veloces para ese tipo de procesos, en nuestro caso, vamos a aprovechar la enorme potencia de python para cálculo matricial y vamos a parametrizar con la matriz de trades, y no con el payoff dia a dia de la estrategia

Pros:

- Mucho más sencillo de codificar
- Muchísimo más veloz de procesar, lo que nos da mas poder de probar de todo

Contras:

- No tenemos el payoff total dia a día de la estrategia parametrizada
- No podemos usar ratios de riesgo que necesiten el payoff dia a dia como el sharpe ratio

Así que usaremos como vimos una tabla resumen de trades y algún ratio final como el tiempo total comprado, la TEA de la estrategia, o ratios de ganadores/perdedores y cosas así. Recuerden que ya habíamos codeado una función que extraía n resumen de estas métricas:

```
def resumen(trades):

    if len(trades):
        resultado = float(trades.iloc[-1].rendimientoAcumulado-1)
        agg_cant = trades.groupby('resultado').size()
        agg_rend = trades.groupby('resultado').mean()['rendimiento']
        agg_tiempos = trades.groupby('resultado').sum()['dias']
        agg_tiempos_medio = trades.groupby('resultado').mean()['dias']

        r = pd.concat([agg_cant, agg_rend, agg_tiempos, agg_tiempos_medio], axis=1)
        r.columns = ['Cantidad', 'Rendimiento x Trade', 'Dias Total', 'Dias x Trade']
        resumen = r.T

        try: t_win = r['Dias Total']['Ganador']
        except: t_win = 0

        try: t_loss = r['Dias Total']['Perdedor']
        except: t_loss = 0

        t = t_win + t_loss
        tea = (resultado +1)**(365/t)-1 if t > 0 else 0

        metricas = {'rendimiento':round(resultado,4), 'dias_in':round(t,4), 'TEA':round(tea,4)}
    else:
        resumen = pd.DataFrame()
        metricas = {'rendimiento':0, 'dias_in':0, 'TEA':0}

    return resumen, metricas
```

Y finalmente obtenía algo así:

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                     trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

trades = getTrades(actions)
resumen(trades)

(resultado      Ganador    Perdedor
Cantidad        7.000000   4.000000
Rendimiento x Trade  0.922483  -0.074515
Días Total     1606.000000  692.000000
Días x Trade   229.428571  173.000000,
{'rendimiento': 29.8979, 'dias_in': 2298, 'TEA': 0.7245})
```

Obviamente pueden agregarle al resumen todo tipo de ratio como los que vimos o cualquier otro para parametrizar en función de ello, este ejemplo que hago es mas pedagógico para ir viendo cómo proceder en términos estratégicos en una parametrización

1º Paso: Performance

Como anticipé unas páginas atrás, no voy a aplicar event-driven para parametrizar en este ejemplito, la idea es pensar antes de codear, el tema es que iterar día a día en python insume un tiempo de cómputo que realizarlo una vez no pasa nada son décimas de segundo, pero la parametrización requerirá probar miles o millones de combinaciones diferentes, con lo cual el tiempo de cómputo en esos milisegundo se pone crucial.

Así que vamos a tener que replantear ese último fragmento de código antes, por suerte lo tenemos todo separado en funciones sueltas, así que veamos como sería el proceso, primero midamos el tiempo en procesar todo:

```
timeit
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                     trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

trades = getTrades(actions)
resultados = resumen(trades)
```

1.08 s ± 96.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

Obviamente que cae de maduro que el getData() podríamos aislarlo ya que la parametrización de todo lo que podemos tomar como variable queda afuera de ese "data" original descargado

Es demasiado obvio, pero bueno, si hubiéramos puesto en la misma función el getData() del cálculo de los features, no podríamos separarlo, y si bien en este caso, al descargarlos de internet es bastante obvio que debe quedar en una función aparte, como verán en el siguiente tomo, cuando ya trabajamos con bases de datos propias no es tan obvio separar ese getData() de la parte de features.

Entonces separando, miren que diferencia:

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')

%%timeit
features = getFeatures(data, n_obv = 100, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                     trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

trades = getTrades(actions)
resultados = resumen(trades)

32.7 ms ± 5.44 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Pasamos de 1.08 segundos a 32.7 milisegundos, bajamos más de 30 veces el tiempo por dejar afuera el hecho de traer los datos. Esto luego se podría optimizar mucho, la verdad no me concentré en eso, sino en dejar un código lo mas pythonico posible y fácil de leer y seguir a fines de ejemplo pedagógico, pero sepan desde ya que pasar de estructuras de pandas a estructuras de numpy ayudaría bastante en mejorar performance.

2º Paso, elegir features y target y “Montecarlear”

Bueno, atenti acá que viene la parte de los “superpoderes”, esta es la parte emocionante, porque después de todo el laburo empieza a tomar mucho sentido el poder que nos da, vamos a empezar a probar exactamente lo mismo que antes pero metido adentro de un FOR que le cambie parámetro por parámetro, empecemos por el primero que aparece por ahí que es la ventana de tiempo para el indicador de volumen que habíamos usado: el OBV

Se los resalto (Como ven pruebo de 2 semanas a 1 año de ventana)

```
l = []
for i in range(10,250):

    features = getFeatures(data, n_obv = i, n_sigma = 40, n_rsi = 15, fast=20, slow = 60)
    actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                         trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

    trades = getTrades(actions)
    resultados_df, resultados_dict = resumen(trades)
    resultados_dict['ventana_OBV'] = i
    l.append(resultados_dict)

results = pd.DataFrame(l)
results
```

	rendimiento	dias_in	TEA	ventana_OBV
0	4.9733	1774	0.4445	10
1	5.1152	1786	0.4478	11
2	5.2365	1790	0.4524	12
...
237	16.5207	2375	0.5528	247
238	16.5207	2375	0.5528	248
239	16.5207	2375	0.5528	249

Como ven obtengo un dataframe con un resultado diferente para cada valor de ventana de días diferente que tomó “i” para el OBV

Esta es una forma rápida y práctica, obviamente hay cosas mucho más performantes con objetos nativos de python, pero la comodidad de usar pandas y sus dataframes me resulta muy tentadora y por ahora no nos jode en cuanto a performance para estos ejemplos.

240 rows × 4 columns

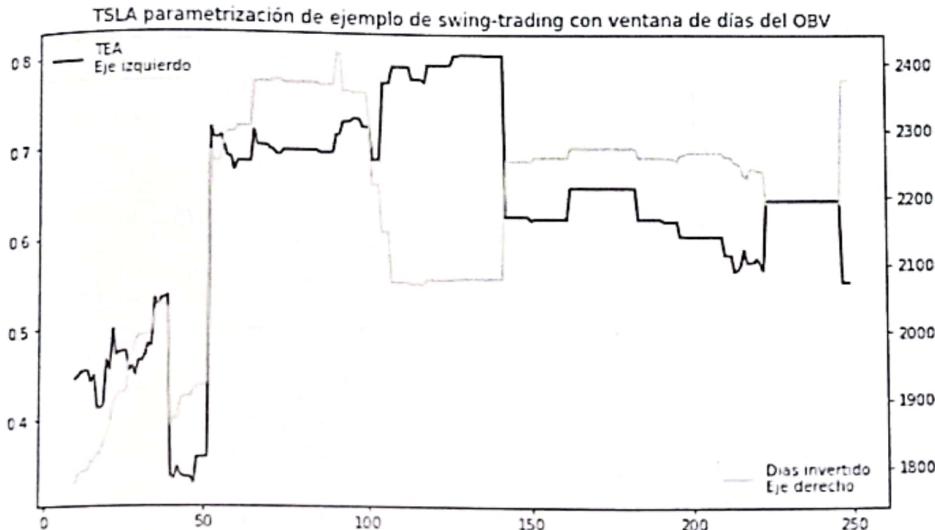
Luego podemos graficar esto en un eje doble donde comparemos la tasa contra la cantidad de tiempo comprado que mantiene el método:

```
fig, ax = plt.subplots(figsize=(10,6))

ax.plot(results.ventana_OBV, results.TEA, color='k', label='TEA \nEje izquierdo')
ax.legend(loc='upper left')
ax.grid(color='gray', alpha=0.2)

ax2 = ax.twinx()
ax2.plot(results.ventana_OBV, results.dias_in, color='silver', label='Días invertido \nEje derecho')
ax2.legend(loc='lower right')

ax.set_title('TSLA parametrización de ejemplo de swing-trading con ventana de días del OBV')
```



Bueno, esto ya nos dispara un montón de reflexiones respecto a la sensibilidad de nuestro método de trading a este indicador, veamos algunos puntos salientes.

- Es muy sensible a cambios, de repente pega saltos la performance por una leve variación del parámetro (Esto no es muy conveniente que digamos para un parámetro)
- Hay una zona clara de óptima performance entre 100 y 135 días aprox.
- Si bien esa zona de 100-135 es óptima para la TEA, en realidad es a costa de estar menos tiempo invertido, si se busca estar mas tiempo invertido, la zona óptima sería de 45-90 días aprox.

Obviamente lo que se puede empezar a hacer acá es afinar la puntería y empezar a buscar valores mas precisos, pero aun no es momento, debido a que recién estamos probando sensibilidad de un solo parámetro, veremos mas adelante como esto empieza a tomar vida propia cuando empezamos a jugar con mas de un parámetro a la vez, así que vamos viendo primero como "pantallazos" de la sensibilidad a parámetros, básicamente vamos a buscar:

- Una zona óptima clara (no un valor sino una zona de valores similares)
- Una zona no tan sensible
- Buscaremos coherencia, es decir si apuntamos a maximizar tasa y reducir tiempo invertido, usaremos el mismo criterio en todos los parámetros, o viceversa.

Probemos ahora con la ventana del RSI (la ventana del sigma me la salteo porque no varia mucho ese parámetro los resultados)

Hacemos exactamente lo mismo que antes, pero ahora dejamos fijo todo el resto y solo vario modificando el parámetro de la ventana de días para construir el RSI:

```
l = []
for i in range(5,120):

    features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = i, fast=20, slow = 60)
    actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                          trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

    trades = getTrades(actions)
    resultados_df, resultados_dict = resumen(trades)
    resultados_dict['ventana_RSI'] = i
    l.append(resultados_dict)

results = pd.DataFrame(l)
results
```

	rendimiento	dias_in	TEA	ventana_RSI
0	20.1139	2714	0.5071	5
1	21.0241	2699	0.5192	6
2	21.8261	2683	0.5304	7
...
112	-0.6297	243	-0.7751	117
113	-0.6297	243	-0.7751	118
114	-0.6297	243	-0.7751	119

115 rows × 4 columns

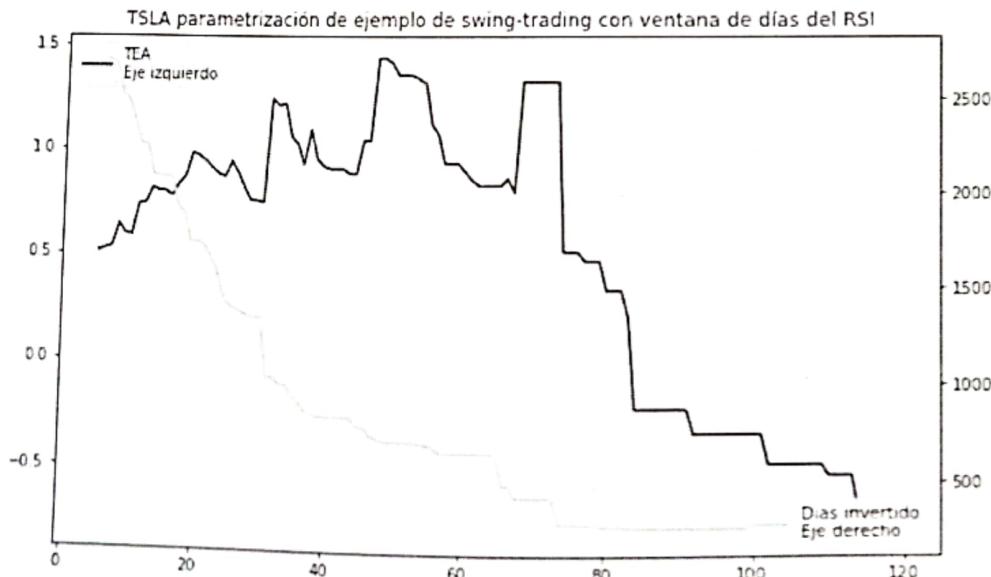
Y ahora grafiquemos esta sensibilidad:

```
fig, ax = plt.subplots(figsize=(10,6))

ax.plot(results.ventana_RSI, results.TEA, color='k', label='TEA \nEje izquierdo')
ax.legend(loc='upper left')
ax.grid(color='gray', alpha=0.2)

ax2 = ax.twinx()
ax2.plot(results.ventana_RSI, results.dias_in, color='silver', label='Dias invertido \nEje derecho')
ax2.legend(loc='lower right')

ax.set_title('TSLA parametrización de ejemplo de swing-trading con ventana de días del RSI')
```



Como ven en el caso del RSI, en lugar de 15 días, podríamos tomar un rango de 45-55 días y casi duplicar la tasa, obviamente con mucho menor cantidad de tiempo invertido, mismo criterio que si aumentábamos la ventana del OBV. Sería un criterio de eficiencia del capital.

Montecarlo con 2 variables a la vez

Hasta acá fuimos probando de a una variable (ventana del OBV o ventana del RSI) pero la pregunta es ¿Qué pasa si vamos estudiando la sensibilidad de a mas de una variable a la vez?

Arranquemos por 2 variables, no es nada del otro mundo, pero como verán empieza a tomar fuerza y complicación, vamos a probar los parámetros del cruce de medias, que son 2 parámetros la media rápida y la lenta, y vemos que pasa si queremos modificar los dos a la vez e ir probando.

Básicamente se resuelve con un for adentro de otro (también se puede hacer un solo FOR que recorra las tuplas prearmadas de cruces, pero no cambia tanto)

```
l = []
for fast in range(5,40):
    for slow in range(int(fast*1.4), int(fast*2) ):
        print(f'Cruce: {fast}-{slow} ', end='\r')
        features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 50, fast=fast, slow=slow)
        actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                             trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

        trades = getTrades(actions)
        resultados_df, resultados_dict = resumen(trades)
        resultados_dict['fast'] = fast
        resultados_dict['slow'] = slow
        l.append(resultados_dict)

results = pd.DataFrame(l)
results
```

	rendimiento	dias_in	TEA	fast	slow
0	6.5095	551	2.8022	5	7
1	4.8992	515	2.5180	5	8
2	4.8992	515	2.5180	5	9
3	8.3356	621	2.7172	6	8
4	6.2241	571	2.5396	6	9
...
471	4.3712	688	1.4396	39	73
472	4.2831	689	1.4152	39	74
473	4.2831	689	1.4152	39	75
474	4.2831	689	1.4152	39	76
475	4.2400	692	1.3956	39	77

476 rows × 5 columns

Por ahora venimos bien, el tema es que cuando queramos graficar esto, va a ser un poco mas complicado montar el EJE X, ya que antes lo hacíamos con el parámetro a estudiar su sensibilidad, pero ahora tenemos dos parámetros y un solo eje X, así que usemos de eje X el número de simulación simplemente y veamos que sale

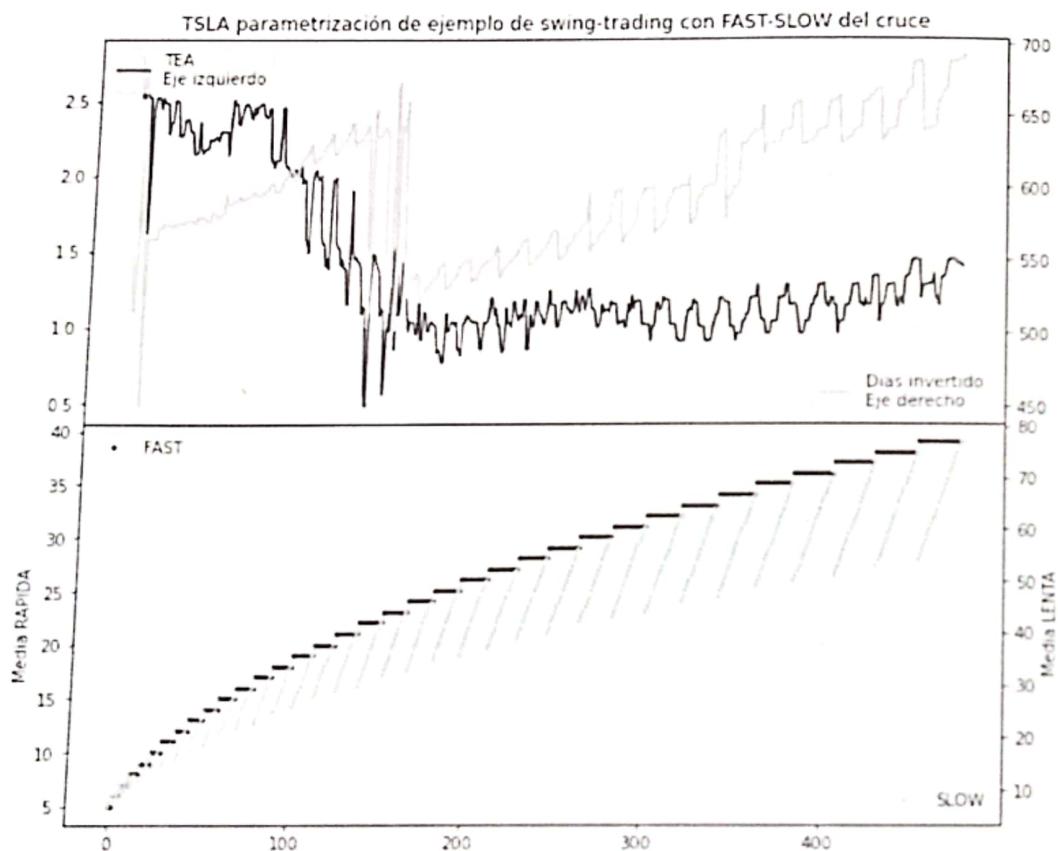
Va gráfico entonces:

```
fig, ax = plt.subplots(figsize=(10,9), nrows=2)

ax[0].plot(results.TEA, color='k', label='TEA \nEje izquierdo')
ax[0].legend(loc='upper left')
ax[0].grid(color='gray', alpha=0.2)
ax2 = {}
ax2[0] = ax[0].twinx()
ax2[0].plot(results.dias_in, color='silver', label='Días invertido \nEje derecho')
ax2[0].legend(loc='lower right')
ax[0].set_title('TSLA parametrización de ejemplo de swing-trading con FAST-SLOW del cruce')

ax[1].plot(results.fast, color='k', lw=0, marker='o', markersize='2', label='FAST')
ax[1].set_ylabel('Media RÁPIDA')
ax[1].legend(loc='upper left')
ax[1].grid(color='gray', alpha=0.2)
ax2[1] = ax[1].twinx()
ax2[1].plot(results.slow, lw=0, marker='o', markersize='2', color='silver', label='SLOW')
ax2[1].set_ylabel('Media LENTA')
ax2[1].legend(loc='lower right')

plt.subplots_adjust(hspace=0)
```



Con un poco de voluntad lo podemos entender, esa forma de "serrucho" es porque la anidación de ciclos FOR la hicimos media lenta => media rápida, es decir que para cada valor de media lenta, va recorriendo las medias rápidas y calculando la performance del método de trading.

Ahora vamos a replicarlo nuevamente pero con mayor apertura, en lugar de usar un factor multiplicativo constante para la media lenta respecto a la rápida vamos a abrirlo mucho mas para ver como resolvemos el gráfico (que se va a complicar)

Como les decía abrimos el abanico de cruces adicionando hasta 100 días a la media lenta respecto a la rápida:

```

l = []
for fast in range(5,50):
    for slow in range(fast+1, fast+100, 5 ):
        print(f'Cruce: {fast}-{slow}', end='\r')
        features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 50, fast=fast, slow=slow)
        actions = getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
                             trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0)

        trades = getTrades(actions)
        resultados_df, resultados_dict = resumen(trades)
        resultados_dict['fast'] = fast
        resultados_dict['slow'] = slow
        l.append(resultados_dict)

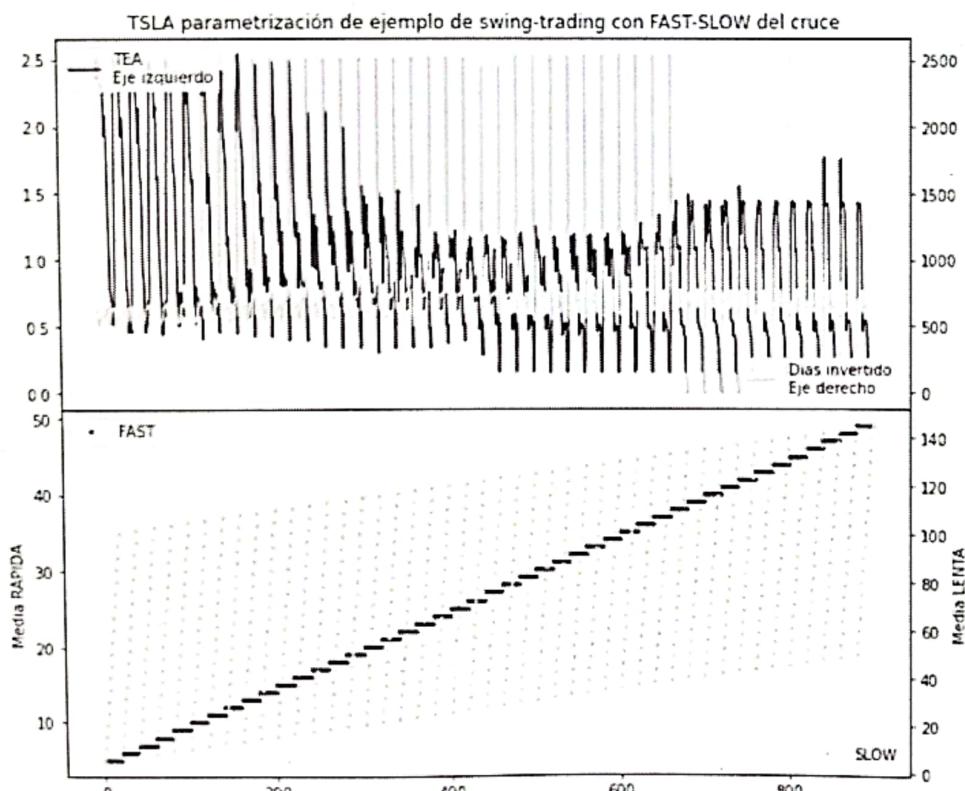
results = pd.DataFrame(l)
results

```

	rendimiento	dias_in	TEA	fast	slow
0	5.6814	556	2.4794	5	6
1	4.5001	514	2.3554	5	11
2	5.9924	564	2.5206	5	16
3	6.0690	572	2.4833	5	21
4	6.0690	572	2.4833	5	26
...
895	0.5488	1259	0.1352	49	125
896	0.7610	1262	0.1778	49	130
897	0.7304	1270	0.1707	49	135
898	0.6944	1271	0.1635	49	140
899	0.6562	1281	0.1546	49	145

900 rows × 5 columns

Esto da el doble de cantidad de combinaciones como ven. Y el gráfico se pone feo (no copio el código porque es el mismo que antes)



En realidad el gráfico se entiende (con buena voluntad) porque se ve claramente la apertura mayor de los cruces y se ve también que de todos modos en los cruces bajos es donde hay mayor performance de TEA, aunque el tema del tiempo invertido es un poco mas complejo de leer (acá al tener esos serruchos, tenemos que leer el chart "en rangos" digamos)

Pero claramente no es un gráfico que me guste, así que vamos a hacer algo más como la gente.

Para ello vamos a importar el interpolador multidimensional de scipy (suena mas cool de lo que es jaja) Lo que me permite este interpolador es pasarle una matriz de 3 dimensiones y generar valores interpolados de la dimensión 3 en función de las otras 2.

```
from scipy import interpolate
import matplotlib.cm as cm
```

El cm es para escalar (mapear por valores) una barra lateral a un color map.

Y luego armamos una función para graficar en 3D (me costó bastante darle forma y que quede bien, así que en breve la subo a pypi y github y para el próximo tomo ya la dejo para usar desde pip install)

```
def graf3D(df, clave_x, clave_y, clave_z, tipo_malla=True, cmap='binary'):

    fig = plt.figure(figsize=(8,8))
    x1 = np.linspace(df[clave_x].min(), df[clave_x].max(), len(df[clave_x].unique()))
    y1 = np.linspace(df[clave_y].min(), df[clave_y].max(), len(df[clave_y].unique()))
    x2, y2 = np.meshgrid(x1, y1)
    Z = interpolate.griddata((df[clave_x], df[clave_y]), df[clave_z], (x2, y2))
    Z[np.isnan(Z)] = df[clave_z].mean()
    ax = plt.axes(projection='3d', alpha=0.2)

    if tipo_malla:
        ax.plot_wireframe(x2, y2, Z, color='tab:blue', lw=1, alpha=0.6)
    else:
        ax.plot_surface(x2, y2, Z, color='tab:blue', lw=1, cmap=cmap, alpha=0.6)
        m = cm.ScalarMappable(cmap=cmap)
        m.set_array(df[clave_z])
        plt.colorbar(m, fraction=0.02, pad=0.1)

    idxmax = df[clave_z].idxmax()
    row_max = df.loc[df.index==idxmax]
    xmax, ymax = row_max[clave_x].values[0], row_max[clave_y].values[0]
    zmax = df[clave_z].max()

    idxmin = df[clave_z].idxmin()
    row_min = df.loc[df.index==idxmin]
    xmin, ymin = row_min[clave_x].values[0], row_min[clave_y].values[0]
    zmin = df[clave_z].min()

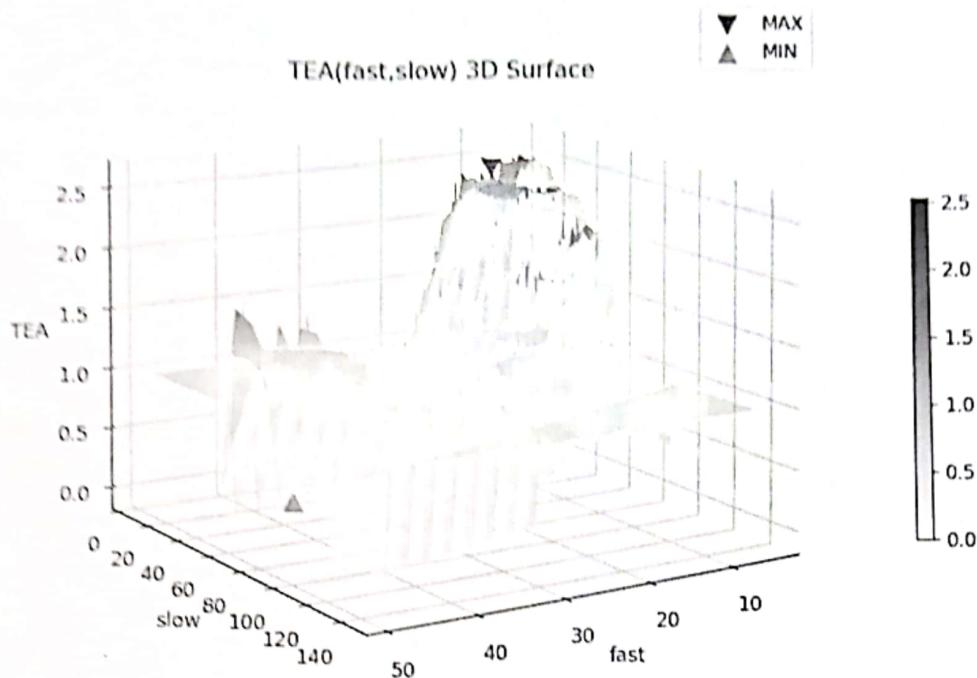
    p = ax.scatter(xmax, ymax, zmax, c='k', marker='v', s=90, label='MAX')
    p = ax.scatter(xmin, ymin, zmin, c='gray', marker='^', s=90, label='MIN')

    ax.set_title(clave_z + "(" + clave_x + "," + clave_y + ") 3D Surface", y=0.95)
    ax.set_xlabel(clave_x)
    ax.set_ylabel(clave_y)
    ax.set_zlabel(clave_z)
    ax.legend()
    plt.show()
```

y luego graficamos simplemente pasando los argumentos de los ejes (nombres de las columnas del dataframe que queremos que sean cada eje)

```
%matplotlib notebook
```

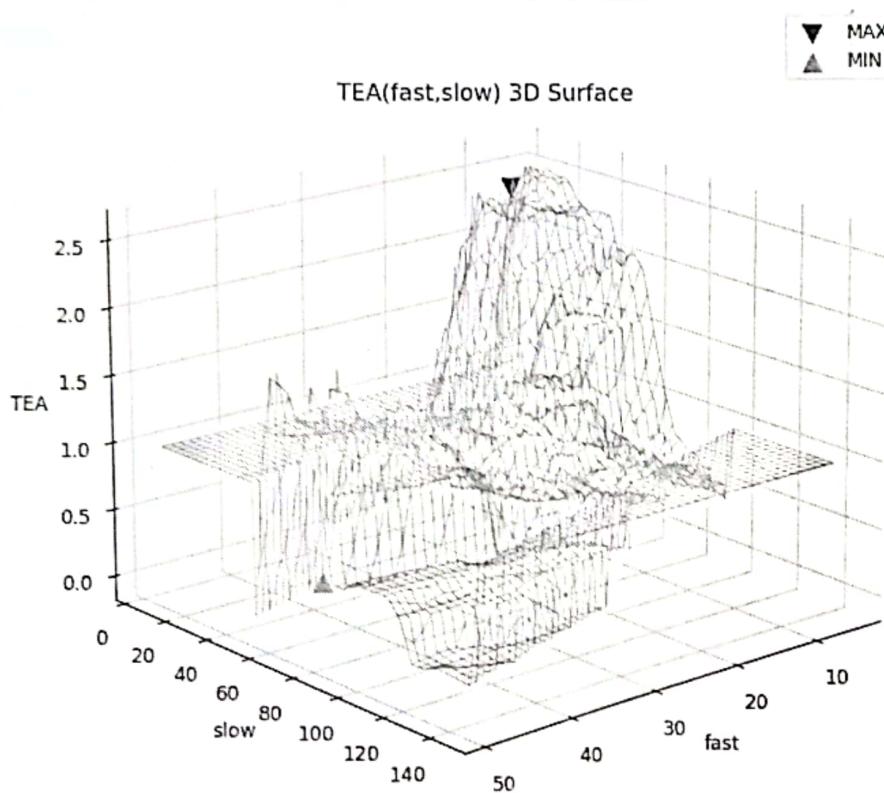
```
graf3D(df=results, clave_x='fast', clave_y='slow', clave_z='TEA', tipo_malla=false)
```



O le pasamos el argumento para graficar tipo malla:

```
%matplotlib notebook
```

```
graf3D(df=results, clave_x='fast', clave_y='slow', clave_z='TEA', tipo_malla=True)
```



Claramente se va entendiendo mejor la idea de los cruces, que en las zonas de cruces bien rápidos es donde mayor TEA nos da, lo mismo con el tiempo invertido sucede aunque en el caso del tiempo la media lenta se extiende mas en lugar de quedar tan encerrada como para maximizar TEA.

Simulación montecarlo múltiple de gatillos

Como se imaginarán si queremos simular todos los gatillos que pusimos (y eso que estamos en un modelito hiper básico mas de muestra que real), se empieza a complicar ya que tenemos 6 gatillos a parametrizar, y no solo debemos encontrar los parámetros óptimos, sino que debemos estudiar la sensibilidad de las variables, tratar de encontrar una forma de probar todo esto a la vez y entender como incide todo, no es tarea fácil, obviamente se puede modelar pero los modelos llevan tiempo elaborarlos y hay que hacer un modelo para cada método de trading digamos.

La solución es nuevamente aplicar montecarlo (fuerza bruta) y entrar a probar con todo miles de combinaciones posibles.

Arranquemos, lo primero que vamos a hacer es armar una función que me permita elegir a la vez un valor aleatorio para cada gatillo.

Sería algo así, por ejemplo:

```
import random

samples = { 'trig_buy_sigma': [i/500 for i in range(20)],
            'trig_buy_rsi': [i for i in range(50,75)],
            'trig_buy_cross': [ -0.05 +i/100 for i in range(20)],
            'trig_sell_rsi': [i for i in range(40,60)],
            'trig_sell_cross': [ -0.1 +i/100 for i in range(20)],
            'trig_sell_obv': [ -2-i/10 for i in range(40)]
        }

def getTriggers():
    triggers = {}
    for key in samples.keys():
        triggers[key] = round(random.choice(samples[key]),4)

    return triggers

getTriggers()

{'trig_buy_sigma': 0.004,
 'trig_buy_rsi': 67,
 'trig_buy_cross': 0.07,
 'trig_sell_rsi': 53,
 'trig_sell_cross': 0.02,
 'trig_sell_obv': -0.1}
```

Notarán que los rangos deben estar en función de la variabilidad de cada feature, es decir, el sigma diario varía centésimas, mientras en el zscore del OBV varía entre -2 y 2 aprox, con lo cual en los simples debo abarcar los rangos propios de cada feature.

Ejemplificando lo anterior:

```

print([round(-2-i/10,2) for i in range(40)])
[-2.0, -1.9, -1.8, -1.7, -1.6, -1.5, -1.4, -1.3, -1.2, -1.1, -1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0.0,
0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9]

data.OBV_osc.quantile(0.05), data.OBV_osc.quantile(0.95)
(-2.0883587817695406, 2.5648546746601855)

```

También podría haber usado el `linspace()` de numpy, queda más técnico, pero más complejo para racionalizar luego los resultados, es mejor trabajar con rangos de números redondos

```

np.linspace(data.OBV_osc.quantile(0.05), data.OBV_osc.quantile(0.95), 20)
array([-2.08835878, -1.84345281, -1.59854684, -1.35364087, -1.1087349 ,
-0.86382892, -0.61892295, -0.37401698, -0.12911101,  0.11579496,
0.36070093,  0.6056069 ,  0.85051287,  1.09541885,  1.34032482,
1.58523079,  1.83013676,  2.07504273,  2.3199487 ,  2.56485467])

```

Lo importante también es que el rango de gatillos sea lógico, porque si pongo de gatillo que espere comprar con un cruce positivo y un RSI mayor a 90 o menor a 10, no va a dar entrada nunca. Por eso se suelen tomar quantiles en realidad mas acotados tipo 0.2 a 0.8 por decir algo, pero depende mucho de cada parámetro, prueben siempre estas cosas.

Bueno, como sea, una vez que tengo la función para generar aleatoriamente los gatillos, me queda

```

%%time
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)

l = []
for sample in range(5000):

    triggers = getTriggers()
    actions = getActions(features, trig_buy_cross = triggers['trig_buy_cross'],
                          trig_buy_rsi = triggers['trig_buy_rsi'],
                          trig_buy_sigma = triggers['trig_buy_sigma'],
                          trig_sell_cross = triggers['trig_sell_cross'],
                          trig_sell_rsi = triggers['trig_sell_rsi'],
                          trig_sell_obv = triggers['trig_sell_obv'])

    trades = getTrades(actions)
    resultados_df, resultados_dict = resumen(trades)

    for key in samples.keys():
        resultados_dict[key] = triggers[key]

    l.append(resultados_dict)

results = pd.DataFrame(l)
results

```

Wall time: 1min 24s

	rendimiento	dias_in	TEA	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_obv
0	0.0000	0	0.0000	0.010	55	0.12	42	0.09	0.8
1	7.4184	2111	0.4454	0.010	52	-0.02	51	0.07	-0.2
2	9.0581	1311	0.9016	0.026	73	0.09	57	0.09	1.7
3	18.4508	1018	1.8983	0.036	57	0.02	57	0.01	0.7
4	0.1101	596	0.0660	0.030	60	0.12	48	-0.06	-1.2
..
4995	0.0000	0	0.0000	0.012	51	0.00	57	-0.08	1.9
4996	3.1682	676	1.1614	0.036	53	-0.05	50	0.07	-1.6
4997	0.0000	0	0.0000	0.020	50	-0.03	53	-0.05	1.9
4998	-0.1525	609	-0.0944	0.000	68	0.08	43	0.09	-1.2
4999	5.3704	1630	0.5138	0.032	59	-0.03	53	-0.07	-0.4

5000 rows x 9 columns

Como habrán notado en el código, saqué afuera del FOR a la función que trae los "features" ya que acá estoy parametrizando solo los gatillos, es decir que los features es siempre el mismo dataframe, por lo cual dejarlo adentro del FOR sería calcular por demás y en estos pasos en donde abusamos de la fuerza bruta es necesario al menos concentrarse en hacer algo de eficiencia, no obsesionarse, pero si al menos lo básico tenerlo bien.

Fijense que tardé 1 minuto para meter 5.000 pruebas, los Montecarlo suelen ser pesados de tiempos, depende de mil cosas, yo acá no lo hice eficiente, podría mejorarse mucho, pero no viene al caso ahora, lo que vamos a hacer es ir mejorando la puntería de poco si quieren para no mandar un proceso de 1 hora o más de una.

Bueno, analicemos los resultados de nuestro montecarlo entonces, la matriz de 5000 simulaciones de mucho no me sirve, así como está, me interesan en realidad los primeros, no sé, digamos 20 resultados.

```
top20 = results.dropna().sort_values('TEA', ascending=False).head(20)
top20
```

	rendimiento	días_in	TEA	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_oby
2866	20.6119	521	7.6110	0.010	56	0.09	59	0.06	12
3929	10.6029	428	7.0885	0.030	62	0.02	59	0.06	-0.5
3457	13.4841	469	7.0069	0.036	60	0.06	59	0.00	0.2
4838	18.2988	521	6.9545	0.036	60	-0.04	59	-0.01	0.9
2710	9.7170	436	6.2833	0.032	61	0.01	59	0.09	-1.1
4328	11.2732	461	6.2810	0.038	60	0.05	56	0.07	-0.3
4946	6.2238	364	6.2631	0.012	68	0.10	59	0.09	1.1
1726	11.7914	477	6.0309	0.032	62	-0.02	55	0.08	-1.7
2552	9.0316	443	5.6843	0.030	53	0.13	58	0.07	1.4
3107	26.4305	639	5.6302	0.038	56	-0.04	58	0.02	-0.6
1578	9.7805	459	5.6246	0.038	62	-0.02	56	0.09	-0.9
3125	13.4709	517	5.5964	0.032	59	0.04	57	0.03	-1.6
4896	4.6465	336	5.5564	0.036	69	-0.02	57	-0.01	0.2
238	4.4243	329	5.5267	0.036	69	-0.04	59	0.02	-1.8
3739	22.0806	618	5.3849	0.032	50	0.07	56	0.05	-0.7
4804	12.6894	516	5.3656	0.038	59	0.04	54	0.04	0.8
1973	12.0582	507	5.3584	0.032	61	-0.04	55	0.06	-0.2
145	12.0686	509	5.3160	0.036	59	-0.04	57	0.06	0.4
2159	4.2040	327	5.3035	0.018	70	-0.04	59	0.01	0.6
4054	11.6331	503	5.2995	0.034	59	0.05	55	0.05	0.0

Bien, ahí tenemos las 20 mejores simulaciones de las 5.000, asumiendo que lo mejor es mayor TEA. Ya con esto, así como está podemos empezar a sacar conclusiones.

Mas allá de los valores de las TEA, o los tiempos adentro, lo que estamos investigando acá es la parametrización de los gatillos, pero esta tabla así no nos dice mucho, vean la dispersión enorme que hay de los valores, así que vamos a tener que empezar a trabajar un poco esto para que tenga sentido.

Empecemos por entender la media y desvío estándar de cada uno de los gatillos en este top20:

```
top20.describe().iloc[1:3, 3:]
```

	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_obv
mean	0.030800	59.750000	0.015500	57.400000	0.047000	0.040000
std	0.008089	4.609772	0.059159	1.602629	0.032298	0.904026

Veamos uno por uno:

- Gatillo compra del sigma: Bien, bastante cercano a 0 y con un +/- bajo, es decir 0.0308 ± 0.008 me deja en un rango desde 0.023 a 0.038, es razonablemente acotado.
- Gatillo de compra RSI: ahí tenemos 59.75 ± 4.6 , o sea de 55.15 a 62.35
- Gatillo de compra en cruce: 0.0155 ± 0.059 (acá el desvío es mas grande que la media, nos deja un rango desde valores negativos hasta positivos, igual es esperable esto porque la media es muy cercana a 0)
- Gatillo de venta del RSI: Vean que este está mejor que el de compra porque la varianza es sensiblemente menor (1.6 vs 4.6 de desvío en el gatillo de compra para el mismo indicador)
- Gatillo de venta de cruce: También me da mejor que en el gatillo de compra, menos varianza.
- Gatillo de venta del OBV: Este da feo feo, básicamente ese desvío me deja en los mismos quantiles que al inicio, como que no tuvo efecto alguno este gatillo.

Así que bueno, si probamos los resultados con las medias, va a dar bien, pero el tema es que habría que buscar la forma de reducir la varianza de los gatillos, porque si no, es muy sensible a pequeños cambios en los mismos me generen una gran diferencia en los resultados, si bien acá no estamos entrenando ningún modelo, es algo similar al efecto del típico overfitting.

Bueno, antes de ver como solucionamos esto, veamos como nos queda el resultado usando los promedios de esos top20:

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
actions = getActions(features, trig_buy_cross = 0.0155,
                      trig_buy_rsi = 59.75,
                      trig_buy_sigma = 0.0308,
                      trig_sell_cross = 0.047,
                      trig_sell_rsi = 57.4,
                      trig_sell_obv = 0.04)

trades = getTrades(actions)
resultados_df, resultados_dict = resumen(trades)
resultados_dict

{'rendimiento': 15.2125, 'dias_in': 534, 'TEA': 5.7136}
```

Nada mal ehh.. ahora pasemos a una segunda vuelta de tuerca para ver como afinamos un poco para obtener un rango de top20 mas acotado, o mas coherente los gatillos entre sí, es decir con menos varianza para obtener así una parametrización más homogénea y confiable.

Habíamos partido de simples medio arbitrarios, yo les mencioné de usar quantiles o bien "a ojo" arrancar el primer montecarlo con "valores razonables" para cada gatillo.

Iterando sobre la parametrización anterior

Bien, pero ahora una vez que ya hicimos un primer montecarlo, ya tenemos, no solo un valor medio de los gatillos sino un desvío estándar, con lo cual podríamos samplear usando esto ahora

Veamos cómo sería el código:

```
from scipy import stats

keys = top20.iloc[:, 3:].columns
samples = {}
for k in keys:
    samples[k] = stats.norm(top20[k].mean(), top20[k].std()).rvs(400)

def getTriggers():
    triggers = {}
    for key in samples.keys():
        triggers[key] = round(random.choice(samples[key]),4)

    return triggers

getTriggers()

{'trig_buy_sigma': 0.0289,
 'trig_buy_rsi': 58.6956,
 'trig_buy_cross': 0.0173,
 'trig_sell_rsi': 57.1193,
 'trig_sell_cross': -0.0106,
 'trig_sell_obv': -1.1056}
```

Como ven a diferencia del modelo anterior, en donde armaba los "samples" a mano y usaba random para elegir de ese rango armado a mano, en este caso, lo que hago es modelar una normal con mu y sigma que saco de la tabla del top20 del montecarlo anterior para cada gatillo. Luego a ese modelado normal le genero un muestreo de 400 muestras (por poner un número) y sampleo desde ahí.

El sampleo de las 400 muestras lo genero con el método rvs() que vimos en el t[6]

Vuelvo a correr el montecarlo (lo único que cambió es un nuevo diccio de samples).

```
%time
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)

l = []
for sample in range(5000):
    triggers = getTriggers()
    actions = getActions(features, trig_buy_cross = triggers['trig_buy_cross'],
                          trig_buy_rsi = triggers['trig_buy_rsi'],
                          trig_buy_sigma = triggers['trig_buy_sigma'],
                          trig_sell_cross = triggers['trig_sell_cross'],
                          trig_sell_rsi = triggers['trig_sell_rsi'],
                          trig_sell_obv = triggers['trig_sell_obv'])

    trades = getTrades(actions)
    resultados_df, resultados_dict = resumen(trades)

    for key in samples.keys():
        resultados_dict[key] = triggers[key]

    l.append(resultados_dict)

results = pd.DataFrame(l)
results
wall time: 1min 24s
```

Y vuelvo a calcular el top20

```
top20 = results.dropna().sort_values('TEA', ascending=False).head(20)
```

	rendimiento	dias_in	TEA	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_cbv
1524	6.7291	319	9.3800	0.0239	68.2877	-0.0534	59.2249	0.0824	1.0163
2857	17.6449	462	9.0878	0.0321	59.2326	0.0671	58.7463	0.0901	1.1695
4918	15.1446	450	8.5465	0.0276	60.4982	0.0696	57.6575	0.0985	-0.5258
4362	14.3004	443	8.4647	0.0379	61.1571	0.0660	57.7589	0.0760	-0.3381
137	14.4376	446	8.3911	0.0376	60.0335	-0.0275	58.9008	0.0677	-0.0689
2320	15.7367	461	8.3078	0.0364	60.8200	0.0396	57.6293	0.0880	-0.0579
1882	16.7806	471	8.3034	0.0378	59.4039	-0.0001	57.7802	0.0955	-2.6470
378	15.9434	465	8.2193	0.0274	60.7059	0.0257	57.7589	0.1099	-0.5258
3670	12.8200	435	8.0569	0.0374	61.9190	-0.0275	58.9318	0.0677	-1.5508
891	15.0395	460	8.0426	0.0364	61.2577	0.0202	57.6007	0.0885	-0.0813
762	15.6532	469	7.9255	0.0329	60.4982	-0.0440	60.0135	0.0562	0.3093
925	14.5672	458	7.9151	0.0371	59.2954	0.0003	59.0029	0.0764	-1.0247
135	15.7651	471	7.8889	0.0359	59.4786	-0.0213	58.0915	0.1024	0.1623
3493	18.6400	499	7.8285	0.0334	59.4786	-0.0064	57.7252	0.1024	0.1573
4419	13.0251	445	7.7547	0.0279	60.5574	-0.0221	59.2320	0.0716	-0.2035
1496	9.2267	392	7.7134	0.0335	61.6577	0.0060	60.0753	0.1133	-0.0689
4193	14.7875	467	7.6414	0.0334	60.7900	-0.0440	57.6062	0.0682	-0.4524
692	15.1297	472	7.5875	0.0324	60.4930	-0.0443	57.7774	0.0807	-0.5257
2084	12.9713	448	7.5716	0.0372	60.7614	0.0677	57.6627	0.0355	0.8718
3322	14.1586	462	7.5659	0.0302	61.0263	0.0577	57.5569	0.0703	0.2335

Esta vez, ya así nomás, a simple vista, puedo darme cuenta que estoy mejorando acotando mucho mas los RSI, se nota al toque, además se nota también como mejoran las TEA que en el montecarlo anterior rondaba el rango 5 a 7 y ahora está en el rango 7 a 9

Pero veámoslo mejor con la tabla descriptiva:

```
top20.describe().iloc[1:3, 3:]
```

	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_cbv
mean	0.033420	60.868610	0.006465	58.336645	0.081965	-0.237535
std	0.004161	1.912115	0.042695	0.843941	0.019180	0.873843

Bueno como ven bajaron todas las varianzas, es decir queda mas acotado y coherente el rango de gatillos parametrizados, es decir que estas medias son más confiables (menos sensibles) que las medias del montecarlo anterior.

Obviamente, con estos valores podría hacer un tercer montecarlo, y así sucesivamente, pero llegará un momento que ya no mejore mas el tema, y no viene al caso tampoco cocinar los datos para buscar un óptimo ahora, la idea era entender una forma de "Montecarlear" muchos gatillos a la vez de una forma medianamente simple.

A modo comparativo les muestro como quedaba la tabla de las primeras 5000 simulaciones Y la segunda es luego de 5 procesos de usar el mu y sigma del montecarlo anterior, es decir 25.000 simulaciones y se obtiene un resultado mucho mejor (casi el doble de TEA, ROI o como lo quieran llamar) pero sobre todo de mucho menor varianza (les dejo solo las columnas de los gatillos de compra y saco los de venta para que me entre en la misma hoja y pueda comparar una al lado de la otra):

rendimiento	dias_in	TEA	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	rendimiento	dias_in	TEA	trig_buy_sigma	trig_buy_rsi	trig_buy_cross
20.6119	521	7.6110	0.010	56	0.09	13.4275	372	12.7208	0.0273	61.9705	-0.0153
10.6029	428	7.0885	0.030	62	0.02	13.4275	372	12.7208	0.0295	61.9683	0.0025
13.4941	469	7.0069	0.035	60	0.06	13.4275	372	12.7208	0.0316	61.9705	-0.0388
18.2988	521	6.9545	0.035	60	-0.04	13.0456	373	12.2718	0.0312	61.9599	0.0017
9.7170	436	6.2833	0.032	61	0.01	13.0456	373	12.2718	0.0303	61.9599	-0.0265
11.2732	461	6.2810	0.038	60	0.05	13.0456	373	12.2718	0.0315	61.9520	-0.0030
6.2238	364	6.2631	0.012	68	0.10	12.7971	371	12.2237	0.0277	62.0156	-0.0502
11.7914	477	6.0309	0.032	62	-0.02	12.7971	371	12.2237	0.0313	61.9833	-0.0576
9.0316	443	5.6843	0.030	53	0.13	12.7971	371	12.2237	0.0302	62.0156	-0.0835
26.4305	639	5.6302	0.038	56	-0.04	12.5543	370	12.0851	0.0317	62.0237	-0.0247
9.7805	459	5.6246	0.038	62	-0.02	12.5543	370	12.0851	0.0325	62.0548	-0.0035
13.4709	517	5.5964	0.032	59	0.04	12.5543	370	12.0851	0.0283	62.0406	-0.0389
4.6465	336	5.5564	0.036	69	-0.02	12.5543	370	12.0851	0.0314	62.0648	-0.0265
4.4243	329	5.5267	0.036	69	-0.04	12.5543	370	12.0851	0.0296	62.0736	-0.0238
22.0806	618	5.3849	0.032	50	0.07	12.5543	370	12.0851	0.0314	62.1426	-0.0344
12.6894	516	5.3656	0.038	59	0.04	12.5543	370	12.0851	0.0326	62.0406	-0.0270
12.0582	507	5.3584	0.032	61	-0.04	12.5543	370	12.0851	0.0302	62.1050	-0.0518
12.0686	509	5.3160	0.036	59	-0.04	12.5339	372	11.8864	0.0294	62.0082	-0.0519
4.2040	327	5.3035	0.018	70	-0.04	12.5339	372	11.8864	0.0288	62.0140	-0.0590
11.6331	503	5.2995	0.034	59	0.05	12.8156	375	11.8813	0.0314	61.9520	-0.0519

Lo mismo podemos comparar la tablita de medias y desvíos de cada indicador

Primer Montecarlo:

top20.describe().iloc[1:3, 3:]						
	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_obv
mean	0.030800	59.750000	0.015500	57.400000	0.047000	0.040000
std	0.006089	4.609772	0.059159	1.602629	0.032298	0.904026

Luego del 5º Montecarlo

top20.describe().iloc[1:3, 3:]						
	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_obv
mean	0.030395	62.015775	-0.034400	62.280335	0.111390	-0.637285
std	0.001530	0.053537	0.023981	0.239359	0.002148	1.095040

Ejemplo, el gatillo de compra del RSI, inicialmente tenía una media de 59.75 pero con un desvío de 4.60, es decir entre 55.15 y 64.35.. Luego del 5º Montecarlo me queda con una media de 62.01 y un desvío de 0.05 es decir entre 61.96 y 62.06, es decir muchísimo más acotado.

Lo mas importante no es maximizar el ROI o la TEA o lo que fuera de ratio de riesgo que usen, sino minimizar la varianza de los parámetros en las simulaciones.

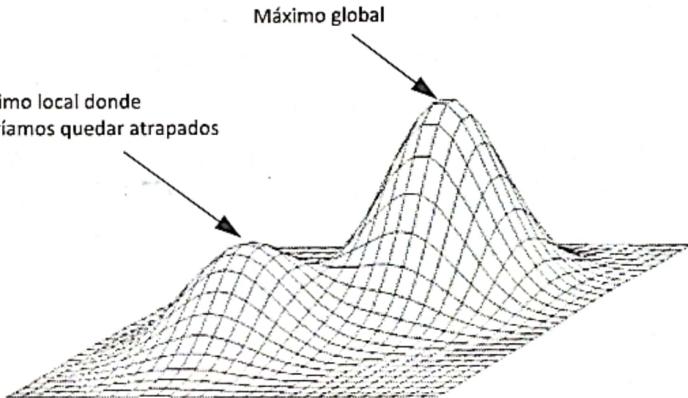
3º Paso y más allá

Esto fue sólo el inicio del análisis de sensibilidad, si seguimos avanzando se abre mucho el camino, de hecho abre todo el camino al tema de inteligencia artificial aplicada a este tipo de usos, que para explicarlo me va a llevar un tomo entero (está planificado para más adelante en esta misma colección)

Pero igualmente hay algunas cositas como "pasos a seguir" para que empiecen a investigar así que se los voy a ir dejando como una especie de bullets, medio desordenados pero ideas a empezar a trabajar mas adelante si avanzan en este tema:

- Algoritmos genéticos, adaptativos y progresivos para optimización. Algo de esto dejé deslizar, un algoritmo genético parte de una generación 0 (podría ser un primer montecarlo) y va evolucionando generación en generación, haciendo que los features más aptos vayan tomando preponderancia en las generaciones siguientes, logrando así que luego de varias generaciones los resultados sean óptimos y haya cada vez menor variabilidad genética ya que los individuos (datos/features) menos aptos no tienen tanto éxito para pasar a las siguientes generaciones como los individuos más aptos.
- Introducción de aleatoriedades bruscas (saltos) para evitar trampas de máximos/mínimos locales: El método iterativo anterior de ir mejorando los pasos del montecarlo previo, tiene la problemática de que puede caer en un máximo local ajustando a pleno eso, pero quedar lejos del máximo global

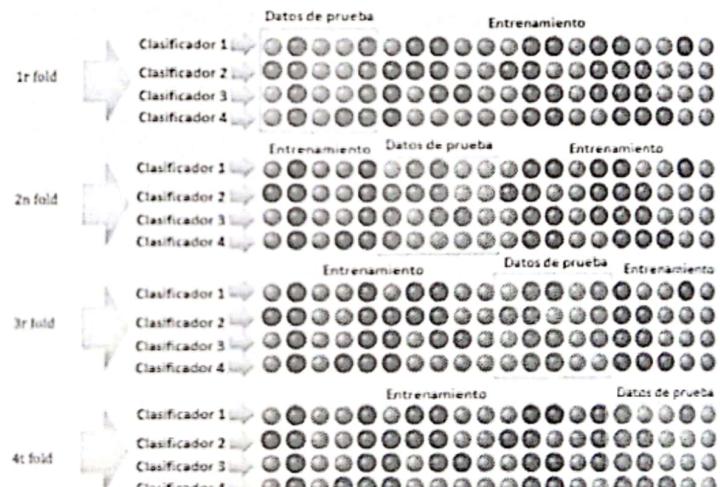
Es un ejemplo con 3D, es decir 2 features y un resultado a optimizar en el eje Z, pero desde ya imaginén la complejidad al usar 6 features como en nuestro ejemplo sencillo



- Algos de machine learning del estilo PCA para reducir dimensionalidad. Si se ponen a pensar u minuto, el ejemplito sobre el cual trabajé todo este tema es super básico y así y todo tiene como 6 variables a parametrizar en los gatillos y otras tanto en los features. Imaginen un modelo de trading mas real con muchas mas cosas a tener en cuenta que un par de indicadores básicos.. Se empieza a poner muy "multidimensional" y eso empieza a ser un problema, con lo cual antes de parametrizar y buscar comprender como afinar las variables, quizás la pregunta sería ¿no son demasiadas variables? Y ahí entran algoritmos de "reducción de dimensionalidad" uno de los mas conocidos es el PCA pero unos 10 modelos diferentes de encarar estos temas con algoritmos específicos, este tema lo cubriremos obviamente en el tomo de IA con la parte de machine learning, pero se los dejo picando para que vayan leyendo.

- Algos de ML supervisados del estilo árboles de decisión para parametrizar ayudados en métricas de performance de accuracy etc. También queda para el tomo de IA, pero básicamente hay algoritmos como los árboles de decisión que me permiten ir "talando" decisiones que no agregan mucho e ir afinando los parámetros sobre los cuales decidimos, es largo de explicar en pocas palabras, pero básicamente este tipo de algoritmos cuentan con toda una serie de herramientas para medir la performance de lo que hacemos, y el grado de "overfitting" o sobreajuste a los datos que tenemos.
- Algos de ML no supervisados para clusterizar datos y poder validar resultados en diferentes clusters. Obviamente también entraremos en este tipo de algoritmos en el tomo de IA, pero básicamente hay algoritmos de aprendizaje no supervisado como la clusterización jerárquica, o el famoso kmeans, el meanshift, el DBScan, los clusterizadores de mezclas gaussianas, en fin hay mucha variedad de algoritmos que me permiten "separar" los datos en conjuntos que tienen parecidos entre sí antes de probar mis backtestings, esto es genial para poder probar o backtestear mis ideas en grupos totalmente diferentes entre sí pero que dentro de ellos tienen cierta homogeneidad, lo que me lleva al siguiente ítem de validación cruzada.
- Hiperparámetros y Cross Validation. Hay parámetros que podríamos dejarlos como constante y luego de parametrizar todo, volver a probar como dan las cosas modificando estos parámetros (hiperparámetros). Asimismo podríamos usar solo una parte de los datos como datos de prueba para parametrizar con ellos, y luego validar con un "resto" de datos con los que no probamos nunca ninguna parametrización (así evitamos overfittear), pero si lo hacemos con una sola partición, podemos overfittear igual o tener resultados no tan buenos, ahora que pasa si repetimos este proceso de separación en datos para parametrizar y datos solo para validar, en varios "sets" de datos diferentes y luego vamos viendo como la parametrización que hicimos es homogénea (o no) entre esos diferentes "sets"

No es complejo hacer esto, pero es un poco engorroso y tedioso de codear simplemente por eso no lo voy a incluir en este tomo. Además que cobra mucho mayor utilidad cuando se utiliza con herramientas más potentes como las de IA, redes neuronales sobre todo.



Bueno, con esas 6 cositas les dejo para entretenerte la que quiera profundizar en mas cosas de parametrización y análisis de sensibilidad. Por ahora soy por terminado este bloque y pasamos a la siguiente etapa de un backtesting que es la de contrastes, validaciones y portabilidad.

Etapa de contrastes y validaciones

Como anticipamos en la breve descripción de etapas de un backtest, la última etapa tiene dos grandes partes

- Análisis de Portabilidad
- Análisis de entornos más realistas

Obviamente el análisis de portabilidad es algo mas acotado, ya que si bien hay infinidad de instrumentos, infinidad de mercados, infinidad de timeframes etc, la verdad que tampoco es infinito y por otro lado son todas cosas que se pueden generalizar, es decir si la idea es ver que tan portable es una idea, por ejemplo, tomemos el ejemplito que venimos usando para operar swings con TSLA, si funciona para TSLA podría funcionar para AAPL o para MSFT, o mejor aún ¿podría funcionar para los 500 del SP500? ¿podría no funcionar para ninguno excepto para TSLA?

Y así hay muchas preguntas a hacerse, pero básicamente siempre son las mismas o rondan en preguntas similares, la idea es ver que tan "generalizable" es nuestra idea o que tan acotada solo al entorno donde la estudiamos es viable..

Ningún extremo es bueno, obviamente si solo sirve para TSLA tema decir que seguramente está overfiteado el estudio de sensibilidad, ahora si sirve para 500 de las 500 del SP500, seguramente algo mal hicimos porque es casi imposible que algo funcione para absolutamente todo ya sea high beta, low beta, small cap, blue chips, value, growth, muy volátiles y poco volátiles etc.. siempre hay características que condicionan en que entorno sirve nuestro método

Luego viene el estudio de los entornos, los criterios de realismo, y aquí es donde aparecen los "finos" o imponderables que nunca se tienen en cuenta al principio (y en realidad esta bien que así sea porque de otro modo sería complicar demasiado en una etapa en la que se ven cosas mas burdas mas a groso modo las ideas digamos) Y como se imaginaran esta etapa es la mas compleja, pero no por eso iba a dejar al menos de mencionarla y ver algunas cositas de las que mas ruido suelen hacer al menos, así que empecemos a desarmar estas dos grandes subetapas de la parte final de un backtest.

Estudio de portabilidad

Como anticipamos la idea es testear como funciona nuestra estrategia para:

- Diferentes acciones
- Diferentes mercados (Acciones de otros países por ejemplo)
- Diferentes instrumentos (en futuros, Forex, o con sintéticos de assets, por decir algo)
- Diferentes timeframes (ejemplo velas horarias, semanales, etc)

Hay mucho para probar, pero la idea es ver que medianamente sirva para mas de un activo (nunca va a servir para todo, pero si entre 500 activos, encontramos que solo sirve en el que probamos y 5 mas y en los otros 495 dan pérdidas, claramente es un problema.. Si vemos que sirve para los 500 también es un problema porque la probabilidad de que eso ocurra sin que sea por un error en nuestro código (lamentablemente) es muy baja, por lo general sirve para activos que tienen cosas en común (alta/baja volatilidad, alto/bajo beta, etc)

Así que empecemos a probar nomás

Primer paso bajarse el historia de precios de muchos activos, en nuestro caso para no salirse mucho de la norma vamos a usar los 500 del SP500, y para ello voy a usar un repo de github mío en el que subí el historial de las 500 del SP500 por fecha desde hace varios años atrás (hice esto porque las APIs públicas que tienen esta data por lo general son endpoints pagos)

El link al repo es este (lo hice para mis alumnos de la Ucema)

- <https://github.com/gauss314/ucema>

El link al histórico de los componentes en formato csv:

- https://raw.githubusercontent.com/gauss314/ucema/main/sp500_historical.csv

Así que bajemos la data:

```
url = 'https://raw.githubusercontent.com/gauss314/ucema/main/sp500_historical.csv'  
r = pd.read_csv(url)  
r
```

	date	tickers
0	1996-01-02	AAL,AAMRQ,AAPL,ABI,ABS,ABT,ABX,ACKH,ACV,ADM,AD...
1	1996-01-03	AAL,AAMRQ,AAPL,ABI,ABS,ABT,ABX,ACKH,ACV,ADM,AD...
2	1996-01-04	AAL,AAMRQ,AAPL,ABI,ABS,ABT,ABX,ACKH,ACV,ADM,AD...
3	1996-01-10	AAL,AAMRQ,AAPL,ABI,ABS,ABT,ABX,ACKH,ACV,ADM,AD...
4	1996-01-11	AAL,AAMRQ,AAPL,ABI,ABS,ABT,ABX,ACKH,ACV,ADM,AD...
...
2633	2021-04-20	A,AAL,AAP,AAPL,ABBV,ABC,ABMD,ABT,ACN,ADBE,ADI,...
2634	2021-05-14	A,AAL,AAP,AAPL,ABBV,ABC,ABMD,ABT,ACN,ADBE,ADI,...
2635	2021-06-04	A,AAL,AAP,AAPL,ABBV,ABC,ABMD,ABT,ACN,ADBE,ADI,...
2636	2021-07-21	A,AAL,AAP,AAPL,ABBV,ABC,ABMD,ABT,ACN,ADBE,ADI,...
2637	2021-08-03	A,AAL,AAP,AAPL,ABBV,ABC,ABMD,ABT,ACN,ADBE,ADI,...

2638 rows × 2 columns

Como ven, me trae un dataframe con la columna de la fecha y la columna de los componentes, sencillo Ahora lo que tenemos que hacer es filtrar los componentes a determinada fecha (antes o después de)

por ejemplo, para traer los componentes que había antes del 1ro de enero del 2015:

```
url = 'https://raw.githubusercontent.com/gauss314/ucema/main/sp500_historical.csv'
r = pd.read_csv(url)
components = r.loc[r['date'] < '2015-01-01']
components.iloc[-1].tickers
```

'AABA,AAPL,ABBV,ABC,ABT,ACN,ADBE,ADI,ADM,ADP,ADS,ADSK,ADT,AEE,AEP,AES,AET,AFL,AGN,AIG,AIV,AIZ,AKAM,ALL,ALLE,ALTR,ALXN,AMAT,AME,AMG,AMGN,AMP,AMT,AMZN,AN,ANDV,ANTM,AON,APA,APC,APD,APH,APTV,ARG,ARNC,ATI,AVB,AVGO,AVP,AVY,AXP,AZO,BA,BAC,BAX,BBBY,BBT,BBY,BCR,BDX,BEN,BF,B,BHGE,BIIB,BK,BKNG,BLK,BLL,BMY,BRCM,BRK,B,BSX,BWA,BXP,C,CA,CAG,CAH,CAM,CAT,CB,CBR,E,CBS,CCE,CCI,CCL,CELG,CERN,CF,CFN,CHK,CHRW,CT,CINF,CL,CLX,CMA,CMCSA,CME,CMG,CMI,CMS,CNP,CNX,COF,COG,COL,COP,COST,COV,CPB,CRM,CSCO,CSX,CTAS,CTL,CTSH,CTXS,CVC,CVS,CVX,D,DAL,DD,DE,DFS,DG,DGX,DHI,DHR,DIS,DISCA,DISCK,DLTR,DNB,DNR,DO,DOV,DOW,DRJ,DTE,DTV,DUK,DVA,DVN,DXC,EA,EBAY,ECL,ED,EFX,EIX,EL,EMC,EMN,EMR,EOG,EQR,EQT,ES,ESRX,ESS,ESV,ETFC,ETN,ETR,EW,E,XC,EXPD,EXPE,F,FAST,FB,FCX,FDO,FDX,FE,FFIV,FIS,FISV,FITB,FLIR,FLR,FLS,FMC,FOSL,FOXA,FSLR,FTI,FTR,GAS,GD,GE,GGP,GILD,GIS,GLW,GM,GMCR,GME,GNW,GOOG,GOOGL,GPC,GRMN,GS,GT,GW,HAR,HAL,HAS,HBAN,HCBK,HCP,HD,HES,HIG,HOG,HON,HOT,HP,HPQ,HRB,HRL,HRS,HSP,HST,HSV,HUM,IBM,ICE,IFF,INTC,INTU,IP,IPG,IR,IRM,ISRG,ITW,IVZ,JCI,JEC,JEF,JNJ,JNPR,JOY,JPM,JWN,K,KDP,KE,Y,KIM,KLAC,KMB,KMI,KO,KORS,KR,KRFT,KSS,KSU,L,LB,LEG,LEN,LH,LLL,LLTC,LLY,LM,LMT,LNC,LO,LOW,LRCX,LUV,LVLT,LYB,M,MA,MAC,MAR,MAS,MAT,MCD,MCHP,MCK,MCO,MDLZ,MDT,MET,MHK,MJN,MKC,MLM,MMC,MMM,MNKT,MO,MON,MOS,MPC,MRK,MRO,MS,MSFT,MSI,M,TB,MU,MUR,MVW,MYL,NAVI,NBL,NDAQ,NE,NEE,NEM,NFLX,NFX,NI,NKE,NLSN,NOC,NOV,NRG,NSC,NTAP,NTRS,NUE,NVDA,NWL,NWSA,OI,OKE,OMC,ORCL,ORLY,OXY,PAXX,PBCT,PBI,PCAR,PCG,PCL,PCP,PDCO,PEG,PEP,PETM,PFE,PFG,PG,PGR,PH,PHM,PKI,PLD,PLL,PM,PNC,PNR,P,NW,POM,PPG,PPL,PRGO,PRU,PSA,PSX,PVH,PWR,PX,PXD,QCOM,QEP,R,RAI,RCL,REGN,RF,RHI,RHT,RIG,RL,ROK,ROP,ROST,RR,RC,RRG,RTN,SB,UX,SCG,SCHW,SE,SEE,SHW,STAL,STJ,SLB,SNA,SNDK,SNI,SO,SPG,SPGI,SPLS,SRCL,SRE,STI,STJ,STT,STX,STZ,SWK,SWN,SWY,SYK,SYMC,SYV,T,TAP,TDC,TE,TEG,TEL,TGNA,TGT,THC,TIF,TJX,TMK,TMO,TPR,TRIP,TROW,TRV,TS,TSO,TSN,TSS,TWC,TWX,TXN,TXT,UAA,UHS,UNH,UNM,UNP,UPS,URBN,URI,USB,UTX,V,VAR,VFC,VIAB,VLO,VMC,VNO,VRSN,VRTX,VTR,VZ,WAT,WBA,WDC,WEC,WELL,WFM,WHR,WIN,WM,WMB,WM,T,WU,WY,WYND,WYNN,XEC,XEL,XL,XLNX,XOM,XRAY,XRX,XYL,YUM,ZBH,ZION,ZTS'

Como ven, me traer todo en un string separado por comas, con lo cual puedo fácilmente transformar a lista y hacer un slice, y de paso traigo los componentes de antes del año 2000 así ya usamos luego estos y no tenemos el problema del sesgo de supervivencia o sesgo de selección (lo del slice es solo para no tener que imprimir todos en pantalla y ahorrar espacio para cosas más importantes)

```
url = 'https://raw.githubusercontent.com/gauss314/ucema/main/sp500_historical.csv'
r = pd.read_csv(url)

components= r.loc[r['date'] < '2000-01-01']
components = components.iloc[-1].tickers
components = components.split(',')
components[:10]

['AABA', 'AAMRQ', 'AAPL', 'ABI', 'ABS', 'ABT', 'ABX', 'ACKH', 'ACV', 'ADBE']
```

Primeras validaciones pre-portabilidad

Antes de empezar a probar mi modelo para los 500 activos del SP500, tendríamos que hacer algunas validaciones previas, entre las mas comunes en términos generales:

- Que mi modelo/activo no tenga ningún feature difícil de conseguir en otros activos
- Que alguno de los features y sus parámetros estén ligados al subyacente en sí
- Que la ventana de tiempo utilizada esté disponible para todos los activos a chequear portabilidad

A modo general son las 3 cosas básicas en el checklist antes de ver portabilidad, muchas veces (y nuestro ejemplo es un claro caso que puse a propósito para hacer foco en esto) se termina volviendo hacia atrás en este paso del backtest por detectar algún problemita en alguno de estos ítems.

— Antes de seguir leyendo, los invito a que piensen como ejercicio, en cual de estos 3 items tenemos un problema en el modelo planteado por nosotros en la parametrización sobre todo.

Asumiendo que se tomaron el tiempo, y asumiendo que muchos notaron el inconveniente en el segundo de esos ítems, pero otros lectores no lo notaron, paso a explicarlo

En nuestro modelo, en uno de los gatillos de compra, usamos un **valor absoluto** de volatilidad, particularmente dijimos luego de parametrizar que el parámetro óptimo de gatillo era cuando el sigma diario (en una ventana de 40 días) era superior a un 3.03%, y esto es un problema a la hora de ver portabilidad (como les dije lo puse a propósito para tener algo concreto de lo cual agarrarme para explicar este problema)

¿Cuál es el problema?

Bueno, que ese sigma a lo mejor para TSLA es común y me genera una cantidad de gatillos importante, pero seguramente para Coca Cola o para otra empresa de menor "Beta" sea un valor altísimo que hace que no se genere ni un solo gatillo.

¿Cómo lo soluciono?

Empezando todo de nuevo

Jajaja, no es joda, o casi, el tema es que la parametrización que hicimos deja afuera a gran parte de los activos del SP500, con lo cual no lo vamos a poder chequear con una gran parte

¿Otra solución maestro?

Y si.. siempre hay alternativas.

Lo mejor antes de llegar a este punto es pensar bien los features, quizá una alternativa a usar la volatilidad como gatillo, hubiese sido usar un zscore en una ventana pasada de la misma, o un cruce tipo volatilidad en 40 días sobre volatilidad en 100, por decir algo, cosa que el valor de gatillo sea relativo y pueda estar representado en todos los activos de u mercado, y no solo valores que a lo mejor son intrínsecos a este activo (ejemplo TSLA que es mucho más volátil que la media)

Así que nos viene bien este inconveniente para replantearnos algunas cosas en las construcción del modelo, en realidad conviene que sea lo mas "agnóstico" al subyacente posible, lo mismo con respecto a los otros 3 items del checklist ehh! No repaso todos para no dejar tanta sarasa sin un ejemplo concreto, pero ustedes me entienden.

Entonces insisto ¿otra solución a empezar de nuevo la parametrización cambiando los gatillos?

Y si, lo que puedo hacer, asumiendo que mi modelo es para activos "high beta" es filtrar activos high beta para probar la portabilidad, esto tiene pros y contras:

Pros:

Foco: Me concentro en activos parecidos y en una idea de trading que a lo mejor esta basada en la volatilidad y al descartar activos poco volátiles mejoro la performance extendida.

Contras:

No voy a tener ni la mas remota idea como hubiera funcionado en otro tipo de activos simplemente porque no dispararían nunca los gatillos

Otra contra es que achicar la muestra de validaciones siempre es una mala idea, pero como tenemos infinidad de activos, realmente no lo consideramos un problema

¿Otra solución master?

Si, obvio, si quisiéramos backtestear con mas activos, o justamente con activos menos volátiles, podríamos bajar el ajuste de ese gatillo, para asegurarnos que sirva para muchos mas activos, por ejemplo bajarlo del 3% al 2% o al 1.5% por decir algo, obviamente que estaríamos hablando de subóptimos y estaríamos mezclando muchas cosas porque por ejemplo la parametrización con esa restricción de entrada, hubiera dado diferente para los otros parámetros..

¿alguna otra solución intermedia?

Claro, hay para todos los gustos, pero desde el momento que imaginé el ejemplo con este problema adrede para explicar todo esto, que en realidad ya tenía en mente la solución, que es la que voy a decir ahora, creo que es la más "sana" en el sentido que limpia potenciales errores de adaptar medio forzosamente lo que teníamos a los activos

¿En qué consiste entonces la solución?

En simplemente eliminar el feature del problema y solamente volver a correr el último montecarlo con el último ajuste, pero eliminando el feature en cuestión.

¿Y como lo elimino? ¿tengo que escribir nuevamente todo el código de los gatillos?

No hace falta, en matemática se habla mucho de las soluciones triviales, nunca nadie les da bola pero en la práctica conceptualmente son super útiles, acá lo que podemos hacer como analogía es aplicar la trivialidad de poner como condición que el sigma deba ser mayor a 0, sabiendo que siempre es >0 , es como anular ese gatillo, o sea tomamos el último montecarlo pero le "hardcodeamos" eso..

Se acuerdan de esta pate? Bueno luego de esto:

```
from scipy import stats

keys = top20.iloc[:, 3: ].columns
samples = {}
for k in keys:
    samples[k] = stats.norm(top20[k].mean(), top20[k].std()).rvs(400)

def getTriggers():
    triggers = {}
    for key in samples.keys():
        triggers[key] = round(random.choice(samples[key]),4)

    return triggers

getTriggers()
```

Aquí tomaba los mu y sigmas del top20 del montecarlo anterior, lo único que hay que hacer es pisar los valores del gatillo que queremos descartar, en este caso el gatillo de la volatilidad para la compra le ponemos una media de 0 con desvio de 0, es decir que en todas las muestras va a tener un valor de 0 que es lo que va a usar como valor para el gatillo:

```
samples['trig_buy_sigma'] = stats.norm(0, 0).rvs(400)
```

O sea que las 400 muestras aleatorias generadas, van a ser todas = 0

Es eso para dejar todo el resto del código igual y correr la última simulación.. o... la forma mas burda aun que es hardcodeando la simulación (ya que estamos integro el código que toma los simples anteriores y lo corro un par de veces:

```
keys = top20.iloc[:, 3:].columns
samples = {}
for k in keys:
    samples[k] = stats.norm(top20[k].mean(), top20[k].std()).rvs(400)

features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)

l = []
for sample in range(5000):

    triggers = getTriggers()
    actions = getActions(features, trig_buy_cross = triggers['trig_buy_cross'],
                          trig_buy_rsi = triggers['trig_buy_rsi'],
                          trig_buy_sigma = 0,
                          trig_sell_cross = triggers['trig_sell_cross'],
                          trig_sell_rsi = triggers['trig_sell_rsi'],
                          trig_sell_obv = triggers['trig_sell_obv'])

    trades = getTrades(actions)
    resultados_df, resultados_dict = resumen(trades)

    for key in samples.keys():
        resultados_dict[key] = triggers[key]

    l.append(resultados_dict)

results = pd.DataFrame(l)
top20 = results.dropna().sort_values('TEA', ascending=False).head(20)
top20.describe().iloc[1:3, :]
```

Y luego de un par de corridas estabiliza la parametrización en torno a estos valores:

top20.describe().iloc[1:3, :]									
	rendimiento	dias_in	TEA	trig_buy_sigma	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_obv
mean	14.748230	452.750000	8.230070	0.0	59.334565	0.077050	57.664035	0.08175	-0.137315
std	0.154701	1.118034	0.101074	0.0	0.353349	0.000611	0.114231	0.00745	0.821056

Como vemos la TEA baja de 12 a 8 y pico, con lo cual claramente el sigma era un dato interesante para el método, esto nos da la pauta que si queremos mejorar el modelito, una primera mejora va a ser arrancar de cero metiendo un rolling sigma relativo u otro rolling sigma, es decir un "cruce de sigmas" o saltos de volatilidad relativa, que sea agnóstico a la volatilidad propia del activo. Pero se los dejo de ejercicio para que practiquen, ya tienen la idea y e fundamento estadístico.

Antes de seguir les dejo una breve guía de como hubiera hecho yo la segmentación, esto en el caso que hubieran dejado el sigma ese alto y hubieran querido buscar activos que tengan volatilidades similares (no recomiendo este tipo de camino a menos que sea para experimentar y aprender de los datos, porque es muy forzada la lógica)

Segmentación previa (paréntesis)

Como les decía, dejo este apartado por si se les ocurría la idea de "segmentar" es decir buscar que activos del SP500 se adaptaban a mis parámetros de gatillo, es decir buscar activos parecidos a TSLA ya que tenía el problema de tener un gatillo con un valor absoluto (sigma) que de hecho en el caso parametrizado (TSLA) es muy alto y solo se va a encontrar en pocos activos del SP500

Como anticipamos también, vamos a usar activos listados antes del año 2010 (de hecho TSLA ni siquiera estaba listado por entonces) y vamos a probar el método que backtesteamos y parametrizamos con TSLA con todos esos activos volátiles de los que estaban listados en el SP500 para el año 2010

```
url = 'https://raw.githubusercontent.com/gauss314/ucema/main/sp500_historical.csv'
r = pd.read_csv(url)

components= r.loc[r['date'] < '2010-01-01']
components = components.iloc[-1].tickers
components = components.split(',')
data = yf.download(components, start='2010-01-01', end='2020-12-31', auto_adjust=True)['Close']
```

Y con esto lo que vamos a hacer es calcular los quantiles de sigma y buscar los similares a TSLA. El uso de los activos listados previo al año 2010 es para evitar el sesgo de supervivencia dado que el método es un método de swing long, con lo que se favorecerá de entrar en activos a los que les vaya bien en el largo plazo.

También vamos a filtrar de lo que bajamos de yahoo finance, solo aquellos activos que contengan mas de 2000 cotizaciones, para asegurarnos que tenemos activos que no tienen todos NaN por ejemplo q pertenecerían a deslistados o con cambios de ticker desde aquella época o simplemente activos que no fueron guardados en la base de datos de yahoo finance.

Para ello armamos un script sencillo:

```
oks, no_oks = [], []
for c in components:
    if len(data[c].dropna()) > 2000:
        oks.append(data[c])
    else:
        no_oks.append(c)

pd.options.display.max_columns = 10
df = pd.concat(oks, axis=1)

print('Lista de acciones sin datos suficientes: ', no_oks)

df
```

Aquí la salida del script anterior:

Lista de acciones sin datos suficientes:

```
['AABA', 'ACS', 'AGN', 'AKS', 'ALTR', 'APC', 'APOL', 'ARG', 'ARNC', 'AVP', 'BBSI', 'BCR', 'BDK', 'BEAM', 'BF.B', 'BHGE', 'BJS', 'BNZ', 'BRCK', 'BTUQ', 'CAM', 'CES', 'CEG', 'CELG', 'CEPH', 'CFN', 'CHX', 'CPWR', 'CTL', 'CVH', 'DELL', 'DF', 'DNR', 'DNR', 'DO', 'DOW', 'DTV', 'EKOKQ', 'EP', 'ETFC', 'FDO', 'FIIZ', 'FLIR', 'FOX', 'FRX', 'FTR', 'GENZ', 'GR', 'HAR', 'HCBK', 'HCP', 'HNZ', 'HOT', 'HRS', 'HSH', 'HSP', 'JAVA', 'JCP', 'DEC', 'JNS', 'LBT', 'LIFE', 'LLL', 'LLTC', 'LM', 'LO', 'LXK', 'MEE', 'MFE', 'MHS', 'MI', 'MIL', 'MJN', 'MOLX', 'MON', 'MAV', 'MAN', 'MYL', 'NOVL', 'NSM', 'NVLS', 'NYX', 'PCL', 'PCP', 'PGN', 'PLL', 'PTV', 'PX', 'Q', 'QLGC', 'RAI', 'RDC', 'RHT', 'RSHCQ', 'RTN', 'RX', 'S', 'SCG', 'SE', 'SHLD', 'SIAL', 'SNOK', 'SPLS', 'STI', 'STD', 'STR', 'SUNEQ', 'SWY', 'SYMC', 'TE', 'TEG', 'TIF', 'TLAB', 'TMK', 'TSS', 'TWC', 'UTX', 'VAR', 'VIAB', 'WFM', 'WIN', 'WYND', 'XL', 'XTO']
```

	A	AAPL	ABC	ABT	ADBE	...	XRAY	XRX	YUM	ZBH	ZION
Date											
2009-12-31		NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
2010-01-04	20.461843	6.572423	22.173862	19.848604	37.090000	...	32.782581	16.199835	19.993305	55.248112	11.434558
2010-01-05	20.239573	6.593766	22.015856	19.688238	37.700001	...	32.392876	16.218607	19.924923	56.997040	11.837728
2010-01-06	20.167664	6.479064	21.807484	19.797585	37.619999	...	32.606281	16.068436	19.792482	56.978638	12.867097
2010-01-07	20.141512	6.467087	21.457769	19.951590	36.889999	...	33.033103	16.143522	19.776783	59.285740	14.308212
...
2020-12-23	116.631310	130.542862	95.775253	105.814415	496.910004	...	51.436024	21.803762	105.029442	148.061127	42.427315
2020-12-24	115.641251	131.549637	95.509323	108.700706	499.859985	...	51.645313	21.612333	105.767082	149.568817	42.261547
2020-12-28	117.158279	136.254608	95.341896	106.149231	498.950012	...	52.133663	22.282333	107.970154	148.210449	42.281048
2020-12-29	116.561714	134.440399	95.657086	106.681015	502.109985	...	51.077236	21.928190	107.645584	151.754196	41.705727
2020-12-30	116.720787	133.294067	95.499481	106.789337	497.450012	...	51.286526	22.150763	107.704605	151.525253	42.144531

2772 rows x 372 columns

Antes de calcular algún quantil de desvío estándar para filtrar, vamos a ver con TSLA que era nuestro subyacente inicial, que quantil es el 3.03% famoso que me daba como límite para gatillo de compra.

Para ello descargamos los datos de TSLA y calculamos los percentiles de cada valor diario de desvío con esa ventana de 40 días que teníamos de parámetro, y luego filtramos los mayores al valor de gatillo (3.04%) y vemos que percentil de la distribución es..

Veamos mejor esto en código:

```
tsla = yf.download('TSLA', auto_adjust=True)
tsla['sigma'] = tsla.Close.pct_change().rolling(40).std()
tsla['percentil'] = tsla.sigma.rank(pct=True)
tsla.sort_values('percentil').loc[tsla.sigma>0.0304]
```

Date	Open	High	Low	Close	Volume	sigma	percentil
2015-11-19	44.108002	45.237999	44.060001	44.360001	12522000	0.030405	0.489774
2011-06-20	5.258000	5.292000	5.100000	5.202000	7689000	0.030439	0.490127
2012-10-31	5.540000	5.670000	5.474000	5.626000	3876000	0.030440	0.490480
2015-11-18	42.900002	44.276001	42.504002	44.214001	14059500	0.030442	0.490832
2019-08-02	46.270000	47.254002	45.846001	46.868000	30682500	0.030447	0.491185
...
2020-03-27	101.000000	105.160004	98.805000	102.872002	71887000	0.088558	0.998590
2020-03-30	102.052002	103.330002	98.246002	100.426003	59990500	0.088582	0.998942
2020-03-24	95.459999	102.737999	94.800003	101.000000	114476000	0.089393	0.999295
2020-03-25	109.050003	111.400002	102.222000	107.849998	106113500	0.089983	0.999647
2020-03-26	109.477997	112.000000	102.449997	105.632004	86903500	0.089983	1.000000

1448 rows x 7 columns

Por lo visto está bastante cerca de la mediana, es decir el gatillo se activa en el percentil 48.97% de sus valores de volatilidad, así que bastante centrado resultó el modelo luego de la parametrización.

Por las dudas corroboramos con la inversa que es mucho más simple:

```
tsla.sigma.quantile(0.4897)  
0.030415554228239137
```

Ok, comprobamos que el quantil 0.4897 efectivamente es esa volatilidad gatillo.

Entonces, ahora si, tomamos los del SP500 del año 2010, calculamos sus volatilidades, les calculamos a cada uno el quantil 0.4897 y de todos ellos tomamos una cantidad tal que el promedio me de cerca al 0.0304 que era nuestro gatillo (digo el promedio para tener activos más volátiles y activos menos volátiles, pero todos más o menos dentro de esa lógica de volatilidad)

Probé con distintas cantidades, y tomando los 30 mas volátiles me da ese promedio, no es mala cantidad 30, así que vamos a elegir esos 30 más volátiles.

```
desvios_medianas = df.pct_change().rolling(40).std().quantile(.4897)  
desvios_medianas.sort_values().iloc[-30:].mean()  
0.029451848993287306
```

Veamos cuales serían:

```
similares = desvios_medianas.sort_values().iloc[-30:].index.values.tolist()  
print(similares)
```

```
['BBY', 'IGT', 'NVDA', 'WDC', 'CF', 'WYNN', 'GME', 'RRD', 'SII', 'FCX', 'POM', 'SVU', 'CNX', 'RRC', 'ATI', 'MU',  
'GNW', 'SWN', 'THC', 'FSLR', 'PBG', 'NBR', 'ANF', 'X', 'ODP', 'AMD', 'CLF', 'EMC', 'TIE', 'COL']
```

O sea que si hubieran optado por este camino de quedarse si o si con ese sigma absoluto, para probar portabilidad, deberíamos buscar activos que al menos tengan la posibilidad de disparar gatillos de compra, de modo contrario daría para la inmensa mayoría de activos que el método no compra nunca

Mi recomendación es no ir por este camino y buscar estudios de portabilidad mas generales, si quieren pueden segmentar luego del estudio de portabilidad, pero sesgar así la muestra no es la mejor idea

Nota al margen, ojalá muchos hayan estado atentos y hayan detectado el "error", pero para los que no, para hacer esa segmentación incurri en el famoso "Look-Ahead Bias" es decir use datos de 2010 a 2020 para segmentar los mas volátiles, y backtestear desde 2010 a 2020, o sea, hice el filtro con datos que no tenia antes de la fecha de inicio del backtest, con lo cual los activos volátiles en 2010 quizá hubiesen sido otros, obviamente voy haciendo estas cosas a propósito para darles espacio a que se planteen todos los potenciales errores que se suelen cometer a la hora de backtestear y muchas veces se subestiman.

Ahora sí, empezamos la portabilidad

Como vimos un par de páginas atrás, tenemos los parámetros de gatillo optimizados con TSLA

```
top20.describe().iloc[1:3, :]
```

	rendimiento	dias_in	TEA	trig_buy_sigma	trig_buy_rsl	trig_buy_cross	trig_sell_rsl	trig_sell_cross	trig_sell_oby
mean	14.748230	452.750000	8.230070		0.0	59.334565	0.077050	57.664035	0.08175
std	0.154701	1.118034	0.101074		0.0	0.353349	0.000611	0.114231	0.00745

Con lo cual, ahora vamos a probar que tan bien (o mal) anda esto con los otros activos del SP500
Pero antes de esto tenemos que modificar algunas cositas de nuestras funciones, nada del otro mundo pero hay especialmente dos cosas que hay que agregar:

- 1- Referencia al CAGR del Buy&Hold
- 2- Bloques Try/Except para activos que no tengan ningún trade

Veámoslo por separado al por qué necesitamos antes agregar esto a las funciones:

- 1- Referencia al CAGR del Buy&Hold: Esto es muy típico de tener que agregarlo a la hora de ver portabilidad, el tema es que al hacer todas las pruebas con un solo asset, nosotros ya sabemos como le fue a ese asset o lo calculamos una vez como ratio comparativo estrategia vs benchmark y no lo volvemos a calcular mas porque el Buy&Hold siempre va a dar lo mismo por mas que cambie la parametrización (como si fuese una constante), el tema es que en un estudio de portabilidad, a diferencia del de sensibilidad o parametrización, esto deja de ser constante porque cambia de activo en activo. Así que hay que agregar este cálculo a la función de reportes.
- 2- Bloques de excepciones: Otra cosa típica a agregar en esta etapa son varios bloques de excepciones, primero porque vamos a correr esto en muchos activos, dentro de los cuales habrá mas cantidad de datos vacíos, NaN que solo en TSLA pero segundo porque es muy probable encontrar activos muy diferentes que no entren en ningún trade, y probablemente ese tipo de cosas no nos haya pasado nunca probando solo con TSLA, así que es momento de poner estas excepciones para que no se trabe todo el código con un error si aparece un solo activo con alguna dificultad de las mencionadas.

Modificación de funciones

Modificamos entonces

- getActions()
- resumen()

Y dejamos como estaban;

getData() getFeatures() y getTrades()

Pego acá las dos funciones modificadas y las explico en la página siguiente

```
def getActions(features, trig_buy_cross=0, trig_buy_rsi=65, trig_buy_sigma=0.01,
              trig_sell_cross = -0.01, trig_sell_rsi=55, trig_sell_obv = 0):
    gatillos_compra = pd.DataFrame(index = features.index)
    gatillos_compra['cruce'] = np.where(features.cruce > trig_buy_cross, True, False)
    gatillos_compra['rsi'] = np.where(features.rsi > trig_buy_rsi, True, False)
    gatillos_compra['sigma'] = np.where(features.sigma > trig_buy_sigma, True, False)
    mascara_compra = gatillos_compra.all(axis=1)

    gatillos_venta = pd.DataFrame(index = features.index)
    gatillos_venta['cruce'] = np.where(features.cruce < trig_sell_cross, True, False)
    gatillos_venta['rsi'] = np.where(features.rsi < trig_sell_rsi, True, False)
    gatillos_venta['obv'] = np.where(features.OBV_osc > trig_sell_obv, True, False)
    mascara_venta = gatillos_venta.all(axis=1)

    data_aux = data.copy().dropna()
    data_aux['gatillo'] = np.where(mascara_compra, 'compra', np.where(mascara_venta, 'venta', ''))

    actions = data_aux.loc[data_aux.gatillo != ''].copy()

    actions['gatillo'] = np.where(actions.gatillo != actions.gatillo.shift(), actions.gatillo,'')
    actions = actions.loc[actions.gatillo != ''].copy()

    try:
        if actions.iloc[0].loc['gatillo'] == 'venta':
            actions = actions.iloc[1:]

        if actions.iloc[-1].loc['gatillo'] == 'compra':
            actions = actions.iloc[:-1]
    except:
        pass

    return actions
```

```
def resumen(trades, data):
    k = data.iloc[-1].Close / data.iloc[0].Close
    years = (data.index[-1] - data.index[0]).days / 365
    CAGR = k**(1/years)-1

    if len(trades):
        resultado = float(trades.iloc[-1:].rendimientoAcumulado)
        agg_cant = trades.groupby('resultado').size()
        agg_rend = trades.groupby('resultado').mean()['rendimiento']
        agg_tiempos = trades.groupby('resultado').sum()['dias']
        agg_tiempos_medio = trades.groupby('resultado').mean()['dias']

        r = pd.concat([agg_cant,agg_rend, agg_tiempos, agg_tiempos_medio], axis=1)
        r.columns = ['Cantidad', 'Rendimiento x Trade', 'Dias Total', 'Dias x Trade']
        resumen = r.T

        try: t_win = r['Dias Total']['Ganador']
        except: t_win = 0

        try: t_loss = r['Dias Total']['Perdedor']
        except: t_loss = 0

        t = t_win + t_loss
        tea = (resultado +1)**(365/t)-1 if round(t,4) > 0 else 0

        metricas = {'rendimiento':round(resultado,4), 'dias_in':round(t,4),
                    'TEA':round(tea,4), 'CAGR_B&H': CAGR}

    else:
        resumen = pd.DataFrame()
        metricas = {'rendimiento':0, 'dias_in':0, 'TEA':0, 'CAGR_B&H': CAGR}

    return resumen, metricas
```

```
getActions()
```

A esta función le tenemos que agregar un bloque try/except porque nosotros asumíamos que si o si iban a haber trades en la lista de trades que traía getTrades(). Pero cuando pruebe con 500 acciones seguro hay alguna que no traiga ningún trade y si no le pongo una excepción ahí, me va a devolver un error de key error o out of bounds.

```
resumen()
```

A esta función le modifiqué varias cositas, en primer lugar le puse un parámetro de entrada mas que es el data, esto es porque necesita saber el CAGR del Buy&Hold del asset sin tradearlo, y a esta función entraba ya la tabla de trades finales pero me pierdo el inicio y fin de la historia del activo, así que debo tener en principio los datos

Luego obviamente le agrego el cálculo del CAGR y se lo agrego al diccionario de salida.

Y finalmente le corregí el cálculo del resultado que tenía un "-1" de más, no jode en la parametrización porque los resultados eran comparativos

Bueno, con estas modificaciones ya estoy en condiciones de empezar a correr portabilidad, pero antes, vamos probando de un activo suelto:

Primeras pruebas manuales de portabilidad

Arrancamos probando con el mismo TSLA para corroborar que funciona todo ok

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
actions = getActions(features, trig_buy_cross = 0.077, trig_buy_rsi = 59.33, trig_buy_sigma = 0.00,
                     trig_sell_cross = 0.081, trig_sell_rsi = 57.66, trig_sell_obv = -0.137)

trades = getTrades(actions)
resumen(trades, data)

(resultado          Ganador Perdedor
Cantidad           4.000000 1.000000
Rendimiento x Trade 1.215247 -0.029945
Dias Total         446.000000 7.000000
Dias x Trade       111.500000 7.000000,
{'rendimiento': 15.6281, 'dias_in': 453, 'TEA': 8.6312, 'CAGR_B&H': 0.6279})
```

Obviamente ya sabíamos que daba perfecto porque ajustamos todo para esta acción. Y como vemos se basa todo en solo 5 trades, es muy poco.

Pero ahora viene la hora de la verdad, probemos con otros activos y vamos viendo:

```
data = getData(ticker='NFLX', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
actions = getActions(features, trig_buy_cross = 0.077, trig_buy_rsi = 59.33, trig_buy_sigma = 0.00,
                     trig_sell_cross = 0.081, trig_sell_rsi = 57.66, trig_sell_obv = -0.137)

trades = getTrades(actions)
resumen(trades, data)

(resultado          Ganador Perdedor
Cantidad           5.000000 4.000000
Rendimiento x Trade 0.142004 -0.043627
Dias Total         392.000000 81.000000
Dias x Trade       78.400000 20.250000,
{'rendimiento': 0.5096, 'dias_in': 473, 'TEA': 0.3741, 'CAGR_B&H': 0.3533})
```

Con NFLX por ejemplo da casi lo mismo que el Buy&Hold, apenas un poquito mas.

Pero probemos con FB

```
data = getData(ticker='FB', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
actions = getActions(features, trig_buy_cross = 0.077, trig_buy_rsi = 59.33, trig_buy_sigma = 0.00,
                     trig_sell_cross = 0.081, trig_sell_rsi = 57.66, trig_sell_obv = -0.137)

trades = getTrades(actions)
resumen(trades, data)

(resultado      Ganador
Cantidad       1.000000
Rendimiento x Trade 0.281424
dias Total     97.000000
dias x Trade   97.000000,
{'rendimiento': 0.2814, 'dias_in': 97, 'TEA': 1.5424, 'CAGR_B&H': 0.2554})
```

Si miramos solo la TEA esta muy bien, pero pará, es 1 (UN) solo trade, con TSLA teníamos 5, con NFLX 9, pero con FB uno solo, no podemos usar este resultado como parámetro, deberíamos decir que al menos tengamos 3 trades, o 5 idealmente para tomar el resultado como válido.

Y de chusma, probemos con el BTC:

```
data = getData(ticker='BTC-USD', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
actions = getActions(features, trig_buy_cross = 0.077, trig_buy_rsi = 59.33, trig_buy_sigma = 0.00,
                     trig_sell_cross = 0.081, trig_sell_rsi = 57.66, trig_sell_obv = -0.137)

trades = getTrades(actions)
resumen(trades, data)

(resultado      Ganador    Perdedor
Cantidad       6.000000   5.000000
Rendimiento x Trade 0.575742 -0.066624
Dias Total     307.000000  57.000000
Dias x Trade   51.166667  11.400000,
{'rendimiento': 6.5999, 'dias_in': 364, 'TEA': 6.6424, 'CAGR_B&H': 0.9336})
```

Parece que este swing funciona bien para los assets volátiles

Probemos entonces con un low Beta

Fijense, por ejemplo, para KO, ¡me da que no hubiera gatillado nunca!

```
data = getData(ticker='KO', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
actions = getActions(features, trig_buy_cross = 0.077, trig_buy_rsi = 59.33, trig_buy_sigma = 0.00,
                     trig_sell_cross = 0.081, trig_sell_rsi = 57.66, trig_sell_obv = -0.137)

trades = getTrades(actions)
resumen(trades, data)

(Empty DataFrame
 Columns: []
 Index: [],
 {'rendimiento': 0, 'dias_in': 0, 'TEA': 0, 'CAGR_B&H': 0.0858})
```

Y como se estarán imaginando, es hora de empezar a preguntarse si va a tener sentido seguir probando los 500 activos del SP500 así como estamos.

Obviamente que lo vamos a hacer, pero vale la pena la reflexión previa, quizá convenga probarlo con otro tipo de activos

Inconvenientes para escalar la portabilidad

¿Cuál es el problema?

Claramente el problema fue que optimizamos la TEA, con lo cual, es razonable que caigamos en este problemita de encontrar soluciones que minimizan la cantidad de trades (minimizarán las entradas en falso, pero también serán muy exigente para las entradas buenas, esperará mucha confirmación digamos)

Esto está perfecto para tradear manual, pero para backtestear nos mete en este dilema, que llegado este punto, cuando queremos ver como funciona en otros activos, vamos a encontrar que se hubiera operado muy poco el método, pero bueno, si fuimos muy exigentes para entrar en un trade, es razonable que nos pase esto, es parte del juego.

Dicho esto, hay varias formas de arreglar el problema para que no nos pase, por ejemplo les explico una básica

En la parametrización, podríamos haber hecho todo tal cual pero descartar el sampleo de triggers por ejemplo si el resultado final incluía menos de una X cantidad de trades, con esto hubiéramos encontrado una TEA menor porque con mayor cantidad de entradas y salidas (mas variabilidad) no es lo ideal desde un punto de vista de inversión, pero si para el estudio de portabilidad..

¿Se dan cuenta lo que pasa?

Si me empiezo a poner exigente con criterios de backtesteabilidad, me tengo que poner a resignar lógica de mi método de trading, y por el contrario si me pongo muy estricto en maximizar mi método de trading, me termino encerrando en algo que se hace menos backtesteable.

Por eso hay tantos quants que se quejan con los backtests con el típico "ñañaña no sirve, no reflejan la realidad" y cosas por el estilo.. ¡Obvio que no refleja la realidad!

No se trata de predecir el futuro, sino de probar las ideas, afinarlas, ponerlas a prueba con datos del pasado etc etc, pero nada de esto implica que en el futuro esto vaya a funcionar, una vez usé esta frase en clase y me entendieron perfectamente así que la replico acá:

"Un backtest no es la demostración de un teorema"

Así que nos guste o no, todo lo que estamos haciendo es para tener un marco de referencia de prueba de nuestras ideas, pero no es la demostración que aplicando esto en el futuro va a dar los mismo resultados que dio en el pasado, así que empecemos probando esto tal cual como lo pensamos sin obligar a la parametrización a fijar condiciones de cantidad mínima de trades.

Para ello a la salida del resumen final debo agregar un contador de trades simplemente, no voy a modificar las funciones para eso, simplemente le agrego un par de líneas a mi disparador del backtest.

Veamos cómo queda el código:

```
data = getData(ticker='TSLA', data_from='2011-01-01', data_to='2020-12-31')
features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
actions = getActions(features, trig_buy_cross = 0.077, trig_buy_rsi = 59.33, trig_buy_sigma = 0.00,
                      trig_sell_cross = 0.081, trig_sell_rsi = 57.66, trig_sell_obv = -0.137)

trades = getTrades(actions)
resumen_df, resumen_dict = resumen(trades, data)
resumen_dict['Cantidad Trades'] = resumen_df.loc['Cantidad'].sum()

{'rendimiento': 15.6281,
 'dias_in': 453,
 'TEA': 8.6312,
 'CAGR_B&H': 0.6279,
 'Cantidad Trades': 5.0}
```

Ahora lo que me queda hacer es correr esto para 500 activos, que hayan estado listados antes de mi época de inicio de backtest y tomar aquellos que hayan tenido al menos 5 trades para ver como da en general el método o calcular un agregado con todos los trades, así sea uno solo por activo.

Para simplificar esta y tener más data parte voy a traer todo el SP500 al día de hoy 20 años para atrás, no aplico el filtro de antes de tal fecha listados, porque yfinance tiene muy mala data vieja y no es el objetivo de este tomo buscar buena calidad de data, se me irían muchas páginas en explicar algo que es mas tema del tomo que sigue de construcción de bases de datos, así que para hacerlo "así nomas" lo dejo y lo hago bien en el tomo que viene esa parte.

Análisis de Portabilidad

Dicho eso, bajamos entonces los últimos 20 años de data del SP500 (aplico igual un mínimo filtro de pedir al menos 2000 datos no nulos)

```
url = 'https://raw.githubusercontent.com/gauss314/ucema/main/sp500_components.csv'
r = pd.read_csv(url)
components = r.Symbol.tolist()

data_sp500 = yf.download(components, start='2001-01-01', end='2021-10-31',
                        group_by='ticker', auto_adjust=True)

no_oks = []
oks = {}
for c in components:
    if len(data_sp500[c].dropna()) > 2000:
        oks[c] = (data_sp500[c].loc[:, ['Close', 'Volume']])
    else:
        no_oks.append(c)
```

De esta forma me queda en el diccionario "oks" todos los dataframes con las columnas que se que vamos a necesitar (Close y Volume) y con un mínimo filtro de calidad, ya que exigimos al menos 2000 datos no vacíos para tomar como válida la serie

Ahora lo único que me queda por hacer es pasar el código del principio de la página por dentro de un FOR que recorra todos los dfs en "oks" y sacar afuera el getData()

Veamos cómo queda ese código:

```
portabilidad = []
for ticker in oks.keys():
    print(f'Procesando {ticker} ', end='\r')
    data = oks[ticker].copy().dropna()
    features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
    actions = getActions(features, trig_buy_cross = 0.077, trig_buy_rsi = 59.33, trig_buy_sigma = 0.00,
                          trig_sell_cross = 0.081, trig_sell_rsi = 57.66, trig_sell_obv = -0.137)

    if len(actions)>0:
        trades = getTrades(actions)
        resumen_df, resumen_dict = resumen(trades, data)
        resumen_dict['Cantidad Trades'] = resumen_df.loc['Cantidad'].sum()
        resumen_dict['ticker'] = ticker
        portabilidad.append(resumen_dict)

port_df = pd.DataFrame(portabilidad)
port_df
```

Procesando ZTS

	rendimiento	dias_in	TEA	CAGR_B&H	Cantidad Trades	ticker
0	-0.1231	36	-0.7361	0.1899	3.0	AOS
1	0.2587	140	0.8217	0.1945	1.0	ABBV
2	3.5562	1005	0.7346	0.1379	17.0	ABMD
3	-0.0492	169	-0.1032	0.2304	6.0	ATVI
4	0.0171	137	0.0461	0.0998	3.0	ADM
...
333	-0.0186	72	-0.0910	0.1988	1.0	XYL
334	-0.0020	24	-0.0303	0.1769	1.0	YUM
335	0.0885	403	0.0798	0.1741	7.0	ZBRA
336	-0.0155	313	-0.0181	0.0179	5.0	ZION
337	-0.0630	18	-0.7329	0.2578	1.0	ZTS

338 rows × 6 columns

Bien, como ven, de acá puedo sacar un montón de data muy interesante, en principio tengo para cada uno de los activos que tienen algún trade, el resultado del CAGR del Buy&Hold vs el ROI o TEA resultante de los trades en los que haya participado cada activo, la cantidad de trades y la cantidad de días y el rendimiento de cada asset.

Con toda esta data puedo sacar un montón de info, hay mucha tela para cortar acá, así que empecemos de a poco, vamos a empezar calculando agregados

Les tiro el código y luego reflexionamos de todo lo que implica:

```
total_dias_invertido = port_df['dias_in'].sum()
total_rendimiento = (port_df['rendimiento']+1).prod()-1
CAGR_Metodo = (total_rendimiento+1)**(1/(total_dias_invertido/365))-1
CAGR_Metodo

0.08262170338117936
```

Como podrán deducir, mi idea fue estimar un CAGR equivalente aplicando la estrategia a todos los 500 activos juntos

Varias salvedades, primero que muchos de esos trades se hubieran superpuesto, lo que hay que hacer es un análisis del capital con mas detalle día a día para que trade tengo capital libre y para cual no, cuando aparecen muchos trades juntos se divide el capital, esto puede ser una ventaja de diversificación o una desventaja porque aprovecha menos la eficiencia del capital, como sea, podemos partir de la hipótesis fracciones de capital para inversión "unidades de trading" digamos, y asumir que el rendimiento es con una de esas unidades de trading que tiene en cuenta la simultaneidad máxima.

Teniendo esto en cuenta nos da un CAGR de la estrategia de poco mas del 8.26%, que no está mal pero tampoco es algo wow, vean lo que pasa si hubiéramos invertido directamente en el SPY:

```
spy = yf.download('SPY', start='2001-01-01', end='2021-10-31', auto_adjust=True)
k = spy.iloc[-1].Close / spy.iloc[0].Close
years = (spy.index[-1] - spy.index[0]).days / 365
CAGR_SPY = round(k**(1/years)-1, 4)
CAGR_SPY

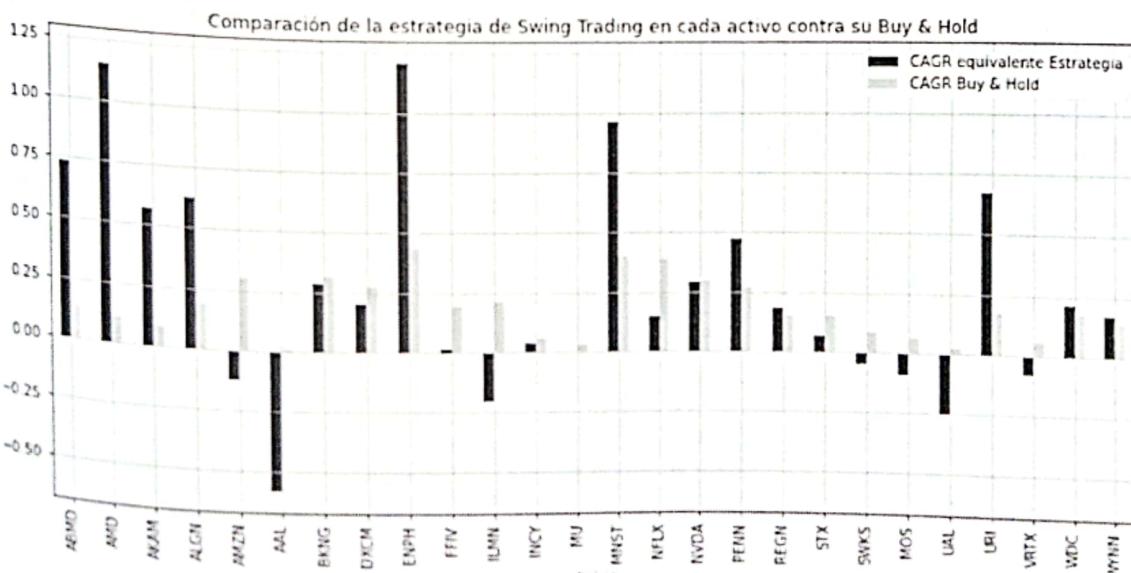
[*****100%*****] 1 of 1 completed
0.0831
```

O sea, toda esta historia para sacar un 8,26% contra un 8.31% que hubiéramos sacado con el SPY

Ahora bien, falta ver todo el tema ratios de riesgo, colas, drawdowns, sharpe etc etc, todo lo que vimos cuando vimos ratios de riesgo, pero se haría infinito el libro si me pusiera a comprar eso acá, se los dejo de ejercicio.

Ahora, también esto es trámposo como comparación porque lo que hay que comparar es el resultado de la estrategia en cada activo en particular contra el Buy&Hold de ese activo, un proxy podría ser a partir de nuestro dataframe anterior, con dos líneas de código tirar algo así:

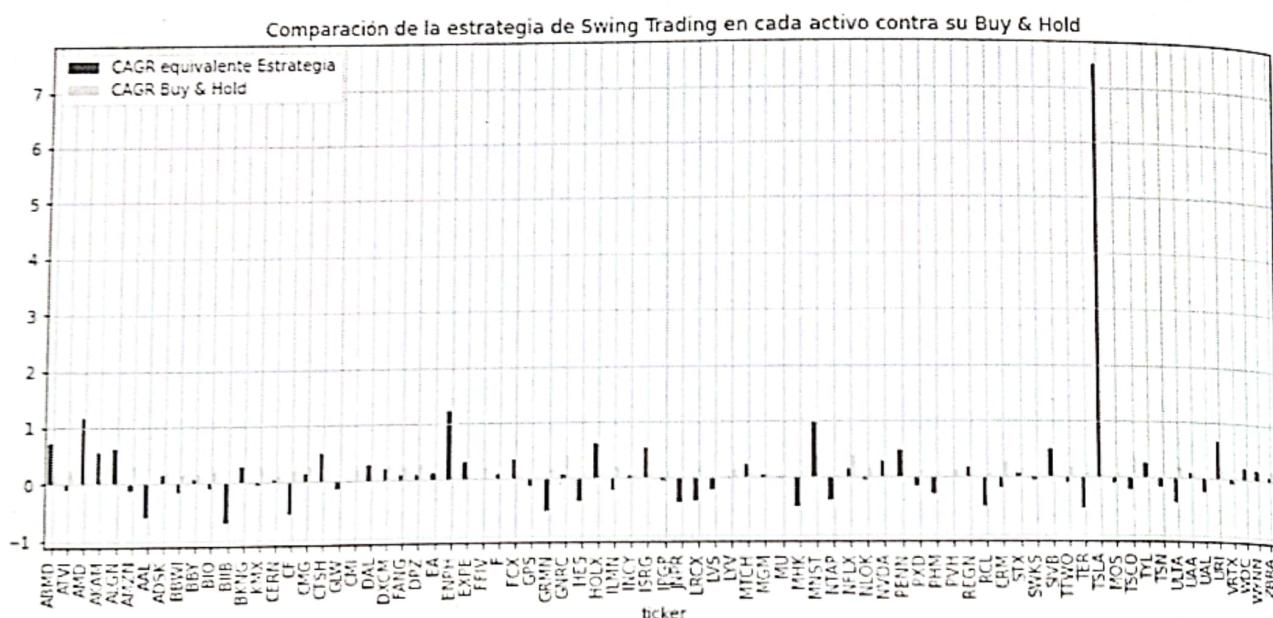
```
results = port_df.loc[port_df['Cantidad Trades'] > 10]
results = results.iloc[:, -4: ].set_index('ticker').iloc[:, :-1]
results.plot(kind='bar', figsize=(15, 6), grid=True)
```



Como ven en el gráfico, filtrando solo los activos de mas de 10 trades (recordemos que puede ser muy confuso evaluar activos con un solo trade por decir algo, además como TSLA tenía menos de 10 trades sacamos del análisis esa acción que estaba sesgada por haber parametrizado con su serie de datos propia obviamente con el resto no va a dar tan bien, era esperable)

Decía, haciendo ese filtro, no se ve tan mal la comparativa, lo que falta estudiar a full en este backtest es como da cada uno de los ratios de riesgo explotando el beneficio de la diversificación, muy probablemente si bien el CAGR es bajo o no justifica la movida, seguramente los ratios de riesgo con esta diversificación en varios trades simultáneos nos dé mucho mejor. Pero tenemos un elefante blanco delante que es a donde quería llegar y todavía no dije nada al respecto, allá vamos..

Ahora vean que pasa si pongo un filtro más amplio q mayor a 10 trades y meto en el juego a TSLA (que fue con la que parametrizamos este bicho), vean por qué, es tan necesario aplicar portabilidad, por mas burdo que sea el esquema de hipótesis planteadas, porque si no hacemos esto, no nos dariamos cuenta de semejante grosería:



Obviamente esa barra gigante es la de TSLA, y obviamente esto nos dice que en cierto modo “overfiteamos” los parámetros para los datos que teníamos, ojo, es lo mismo que hacemos “manualmente” cuando estudiamos un gráfico y los parámetros para ese activo, solo que con python la potencia que tomamos es infinita.

En definitiva, reparar un método de trading (manualmente o con Python) para datos pasados en un solo activo particular, solo va a funcionar en ese pasado con ese activo, parece una obviedad, pero vale la pena recalcarlo. Eso no quita la utilidad de realizar un backtest de una idea de trading, por supuesto q es super útil para entender a donde nos llevaría aplicar la idea, pero hay que tener cuidado extremos con las conclusiones a las que llegamos.

Resumiendo, la parte más rica de hacer un backtest es la que viene ahora de reflexiones

Primeras reflexiones

Les voy a ir dejando reflexiones que me vienen a mi a la mente con los resultados del backtest de nuestro modelito básico de trading, que no necesariamente sean las reflexiones que les vienen a la mente a ustedes ni para nada son las cuestiones mas relevantes del tema hasta ahora antes de pasar a la siguiente etapa, pero no sé, al menos a mi me generaron cierta reacción para compartirles

¿Estuvimos bien al elegir TSLA y parametrizar con esta acción?

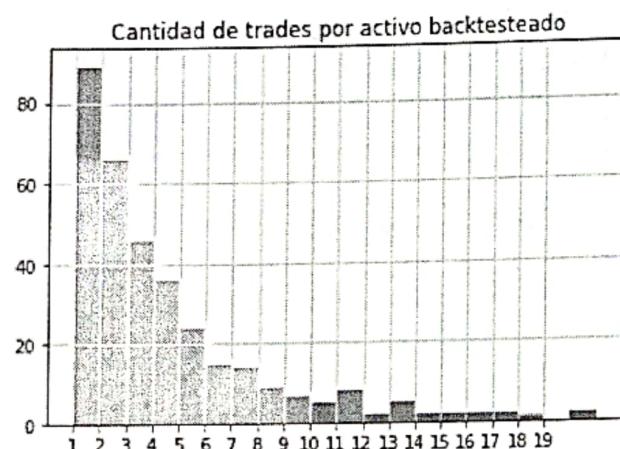
Claramente no. Es decir, está buenísima TSLA para tradear y para holdear por miles de razones, pero justamente es un asset muy poco representativo, mucho mas volátil que la media de activos de similar marketCap y volúmenes, con mucha "vida propia" es decir muy despegada muchas veces del resto, con noticias y eventos muy disruptivos que la desmarcan de las tendencias previas y la colocan de un día para el otro en otra situación totalmente diferente, y como si todo eso fuera poco, con un líder muy mediático que es el centro de la escena todo el maldito tiempo jaja, así que nada, la puse medio a propósito sin saber como iba a dar el tema de la parametrización y la portabilidad, pero imaginándome que seguro pasaba lo que pasó para que quede claro el concepto, si vamos sabiendo de entrada que querremos parametrizar y optimizar y buscar portabilidad con muchos activos, realmente deberíamos elegir activos mas "típicos" o que no se salgan tanto de la media

El problema de backtestear métodos que generan pocos trade

Otra de las reflexiones que quiero traer a la mesa es el "n", es decir la cantidad de trades.

Hay mucha bibliografía de estadística acerca de temas muestrales, representatividad y demás para buenos intervalos de confianza, pero muy poco de eso mismo, pero más relacionado a un fenómeno con tanto nivel de azar como los activos financieros y tanto de correlación y distribuciones con colas tan pesadas.

Lo que nos damos cuenta con el modelo de trading planteado es que el "n" es demasiado bajo, si bien en el total de 500 activos "out of sample" en la parametrización dio un resultado bastante digno a pesar de haber parametrizado con un bicho tan raro como TSLA (esto me deja una buena sensación al menos)



Como les decía el "n", no hay un número mágico, pero con 30 trades por activo es un número bastante piola para empezar a hablar, lo ideal sería que el backtest arroje cientos de trades por activo, con lo cual al ver su portabilidad con 500 activos, tendríamos cantidades superiores a 100mil trades, sería una maravilla, pero con swings del tipo semanales es complicado lograr esas cantidades, aunque es bastante mas probable en timeframes mas bajos y/o con activos que cotizan 24/7.

Resumiendo esto, yo diría que busquemos siempre superar los 20, idealmente 30 trades por activo, sobre todo con el que usen para parametrizar los gatillos. No es que no se pueda con menos, de hecho, en nuestro ejemplito, tenemos muchísimo menos (sólo 5) y parametrizamos con esos igual. El tema es que se corre riesgo muy alto de overfitting a medida que el "n" es más chico, no obstante, es una prueba ácida de aquellas en la portabilidad, es decir, si con un "n" super chico, luego de parametrizar (sabiendo que estamos ajustando mucho al activo elegido) igualmente en la portabilidad me da algo respetable, es un muy buen indicio.

El framework y el ciclo de mejora continua

Mal que mal, a pesar de la cantidad inmensa de temas involucrados en un backtest, el hecho de haber contado con un framework o esqueleto de ideas sobre las cuales trabajar nos facilita y ordena el trabajo, hay miles de cosas mas, de hecho, en este pequeño apartado reflexivo quería puntuar algunas cositas del ciclo de mejora de un backtest

Ojo que llamo ciclo de mejor, no a algo iterativo de optimización, sino a algo más reflexivo

Como dijimos, ya vimos varias cositas a mejorar durante el proceso, les dejo las 3 más relevante:

- Usamos un gatillo referido a un dato intrínseco al subyacente (sigma absoluto diario)
- Elegimos un activo (TSLA) poco representativo del espectro de activos (acciones) en este instrumento
- Usamos una parametrización que optimiza el ROI, TEA o como lo llamen, pero sin tener en cuenta que minimiza mucho la cantidad de trades y eso luego fue un problema en la portabilidad.

Entonces, a partir de esto, lo que podemos hacer es decir, ok, terminado mi backtest, vamos a buscar los ítems mas criticables y veamos como lo podemos atacar para mejorarlo, por ejemplo, veamos como parametrizar de forma similar a como lo hicimos, pero poniendo una cantidad de trades mínima:

```
keys = top20.iloc[:, 3:].columns
samples = {}
for k in keys:
    samples[k] = stats.norm(top20[k].mean(), top20[k].std()).rvs(400)

features = getFeatures(data, n_obv = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)

l = []
for sample in range(5000):

    triggers = getTriggers()
    actions = getActions(features, trig_buy_cross = triggers['trig_buy_cross'],
                           trig_buy_rsi = triggers['trig_buy_rsi'],
                           trig_buy_sigma = 0,
                           trig_sell_cross = triggers['trig_sell_cross'],
                           trig_sell_rsi = triggers['trig_sell_rsi'],
                           trig_sell_obv = triggers['trig_sell_obv'])

    trades = getTrades(actions)

    if len(trades)>20:
        resultados_df, resultados_dict = resumen(trades, data)
        resultados_dict['n'] = len(trades)
        for key in samples.keys():
            resultados_dict[key] = triggers[key]
        l.append(resultados_dict)

results = pd.DataFrame(l)
top20 = results.dropna().sort_values('TEA', ascending=False).head(20)
top20.describe().iloc[1:3, :]
```

Como ven en la parte que resalto del código era cuestión simplemente de agregar una restricción de cantidad de trades en las simulaciones para que la tome como simulación válida, luego en el proceso iterativo de la parametrización, el siguiente paso va a ir afinando igualmente con la misma restricción y así lograremos una parametrización de gatillos que obviamente va a ser sub-óptima en cuanto a ROI pero tendrá un "n" mucho mayor, y unos resultados de portabilidad mucho más confiables, lo hice con solo 2 iteraciones para probar y me da esto:

	rendimiento	dias_in	TEA	CAGR_B&H	n	...	trig_buy_rsi	trig_buy_cross	trig_sell_rsi	trig_sell_cross	trig_sell_ovb
mean	15.798910	577.100000	4.948820	6.279000e-01	18.004805	...	62.104445	-0.035840	60.023829	0.059670	1.095835
std	1.155374	14.728776	0.084325	1.139065e-16	1.542803	...	0.130314	0.011783	0.043229	0.057675	0.089777

2 rows x 11 columns

Luego aplico el mismo código para testear portabilidad pero ajustando esos parámetros:

```

portabilidad = []
for ticker in oks.keys():
    print(f'Procesando {ticker} ', end='\r')
    data = oks[ticker].copy().dropna()
    features = getFeatures(data, n_ovb = 125, n_sigma = 40, n_rsi = 45, fast=15, slow = 30)
    actions = getActions(features, trig_buy_cross = -0.035, trig_buy_rsi = 62.1, trig_buy_sigma = 0.00,
                          trig_sell_cross = 0.085, trig_sell_rsi = 60.02, trig_sell_ovb = 1.1)

    if len(actions)>0:
        trades = getTrades(actions)
        resumen_df, resumen_dict = resumen(trades, data)
        resumen_dict['Cantidad Trades'] = resumen_df.loc['Cantidad'].sum()
        resumen_dict['ticker'] = ticker
        portabilidad.append(resumen_dict)

port_df = pd.DataFrame(portabilidad)
port_df

```

	rendimiento	dias_in	TEA	CAGR_B&H	Cantidad Trades	ticker
0	0.2080	1134	0.0627	0.0811	29.0	MMM
1	0.1681	1536	0.0376	0.1899	34.0	AOS
2	0.0373	1403	0.0096	0.1156	27.0	ABT
3	0.5059	464	0.3799	0.1925	9.0	ABBV
4	12.9549	1928	0.6471	0.1379	30.0	ABMD
...
469	0.0564	1487	0.0136	0.1769	42.0	YUM
470	0.3608	1012	0.1175	0.1741	34.0	ZBRA
471	0.1609	1681	0.0329	0.0850	27.0	ZBH
472	-0.6995	1127	-0.3225	0.0182	25.0	ZION
473	0.1870	463	0.1447	0.2578	15.0	ZTS

474 rows x 6 columns

Y ya puedo ver como me dan mucho mayor cantidad de trades en general

```

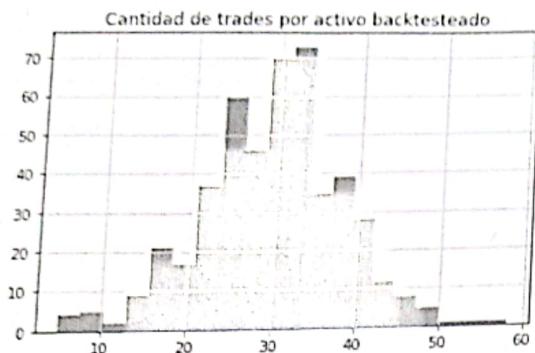
port_df['Cantidad Trades'].mean()

29.873417721518987

```

Y si tiro un histograma de cantidad de trades, me da algo muchísimo más razonable y tendiendo a una distribución normal:

```
plt.hist(port_df['Cantidad Trades'], bins=20)
plt.grid()
plt.title('Cantidad de trades por activo backtesteado')
```



A costa claramente de bajar el CAGR o rendimiento medio:

```
total_dias_invertido = port_df['dias_in'].sum()
total_rendimiento = (port_df['rendimiento']+1).prod()-1
CAGR_Metodo = (total_rendimiento+1)**(1/(total_dias_invertido/365))-1
CAGR_Metodo
0.06010911259303664
```

Recuerden que daba cerca del 8% igual que el SPY en el mismo periodo. Lo que ahora al bajar me obliga a tener que compensar bajando el riesgo para justificar esa baja de rentabilidad.

En fin, quedaría mucho mas para analizar pero ya me entienden la idea estimo.

Sabor semiamargo: "Ladran Sancho, Señal que cabalgamos"

Por último en esta parte reflexiva, no quería dejar de marcar estas tres posibilidades que son las que pueden aparecer al terminar el backtest y mirar los primeros resultados:

- Muy bueno: Si un backtest da muy pero muy bueno que gana un % demasiado elevado de veces, o que le gana por mucho al Buy&Hold y además disminuye el riesgo al mismo tiempo, seguramente es porque tiene algún error metodológico, o un pequeño error en el código, o un error del tipo LookAhead bias, en fin el típico "demasiado bueno para ser real"
- Muy malo: Si un backtest, da demasiado malo, puede ser que la idea era realmente mala, lo cual es bueno o muy positivo porque el hecho de que lo hayamos backtesteado es porque se nos ocurrió, no se hagan los boludos eh, jaja, igual me ha pasado mil veces confieso, pero bueno, justamente para esto uno backtestea, en mi caso me ha evitado hacer muchas cosas realmente estúpidas en el mercado gracias a haberlo backtesteado antes y darme cuenta de que no tenía sentido la idea.

- ¿Quéjese? Ahora lo mejor es justamente cuando el resultado de un backtest "es un gris". Me refiero a casos como este, donde quizás no le ganamos al benchmark pero tampoco estuvimos tan mal, pero al menos le bajamos el riesgo, y nos quedamos en el tintero con muchas cosas a mejorar.
Justamente el "gris" en las conclusiones te deja pensando, te obliga a ver el fino de cada cosa, recalcular entender como incide cada parámetro, cada gatillo, cada momento del mercado, etc,

Entornos más realistas, más reflexiones

Bueno, habiendo repasado ya algunas de las fallas típicas en la parametrización y la portabilidad, falta la prueba más ácida de todas, es la última valla a sortear por el modelo de trading algorítmico, si pasa esta prueba estaría en condiciones de ser testeado en producción, en un ambiente real.

El tema es que acá hay que empezar a considerar algunas cosas dentro de todo sencillas (como las comisiones que son datos de antemano) y otras un poco o bastante más complejas, como los spreads, impuestos, slippage por liquidez por decir alguno, así que veamos lo que deberemos considerar:

Primero enumero los principales ítems, luego los vemos un poco más en detalle:

Variables

- Comisiones
- Derechos de mercado

Semivariables

- Spreads
- Volúmenes y liquidez
- Impuestos

Fijos

- Costos de servicios
- Costos de Infraestructura

Comisiones

Como les decía este es el más sencillo, sin embargo, ojo que no es tan sencillo tampoco si lo queremos considerar realista. Ejemplo, hay mercados en los que un mismo instrumento opera a través de diferentes mercados sobre la misma plataforma, por ejemplo, opciones en el mercado USA, por exchanges como IBKR, puede operar por 3 o 4 mercados diferentes, cobrando comisiones diferentes en función de que subyacente sea, del mismo modo hay ETFs que subsidian costos operativos en sus spots u opciones, y así una variedad muy alta.

¿Cómo solucionamos esta variabilidad?

Fácil, tomando un valor medio, hacer un barrido de las principales opciones, asignarles un porcentaje de probabilidad de operar ese instrumento y después calcular una especie de "comisión media ponderada", pensé en escribirlo con una formulita, pero es al pedo, es decir mucho símbolo que no es necesario y estoy evitando poner.

Mas simple, si tenemos 3 instrumentos con comisiones del 0.3%, 0.2% y 0.1% y la probabilidad de operar esos 3 instrumentos es 20%, 30% y 50% respectivamente, mi comisión media será:

```
import numpy as np

comisiones = np.array([0.003, 0.002, 0.001])
prob = np.array([0.2, 0.3, 0.5])

comision_media = comisiones.dot(prob)
print(f'La comision media es del {comision_media:.2%}')

La comision media es del 0.17%
```

Ahora me preguntarán ¿Cómo sé la probabilidad de operar X o Y instrumento?

Bueno, eso sale de los backtests, si en una simulación tienen 2 instrumentos y uno opera 15 trades y el otro 30, tenemos unas proba de 33.33% y 66.66% respectivamente.

En nuestro ejemplo teníamos muchos instrumentos, la probabilidad de cada uno es simplemente la cantidad de trades de cada uno dividido por los trades totales

Derechos de mercado

En muchos mercados y países estos derechos de mercado son nulos, despreciables o muy bajos, en Argentina lamentablemente no son nada despreciables, operar una acción tiene un derecho de mercado del 0.08%+IVA, con lo cual el trade sería $2 * 0.08\% * 1.21 = 0.194\%$ por comprar y vender una acción, en otros mercados esto es el costo transaccional total o incluso puede ser mucho menor a esto en muchos casos.

En el caso del mercado argentino en opciones tenemos 0.2% de derechos de mercado en cada operación +IVA por supuesto, lo que hace un derecho de mercado del 0.48% para comprar y vender una opción, una locura total.

Como sea, tampoco es lineal ni tan fácil estimar el derecho de mercado, muchos mercados tienen, para ciertos instrumentos, programas de incentivo a market makers que generalmente son los mismos brokers y muchas veces trasladan ese beneficio (en parte) a los minoristas, es bastante complejo meterse en el fino de este tema, pero como la mayoría de las veces es un costo despreciable, nadie le da mucha importancia, pero ojo en mercados donde los costos no son nada despreciables.

A diferencia de las comisiones que dependen exclusivamente del Exchange o bróker, los derechos de mercado dependen del país, mercado, regulaciones etc.

Spreads, volúmenes y liquidez

Bien, hasta ahora venimos bien porque tanto las comisiones como los derechos de mercado son variables, en realidad me podrán decir que no son solo variables porque hay brokers que te cobran por ejemplo USD 1 por operación + 0.1% (digo por poner un ejemplo cualquiera), en ese caso si bien es cierto que no es un costo variable puro, uno puede asumir una operación promedio pongamos un ejemplo de USD 5.000, y tendría por operación $USD\ 1 + 0.001 * 5.000 = 0.12\%$ (estaría prorratoando el fijo para simplificar y asumir un costo variable del 0.12%)

Lo bueno de terminar asumiendo un único costo variable, si bien es una simplificación, es que a nuestro backtest en algún lugar donde calcula el "P&L por trade" simplemente le resto el costo transaccional porcentual y listo! Super sencillo aplicar este criterio de realismo a mi framework de backtesting, pero si quisiera aplicar mezcla de costos fijos y variables habría que entrar a modificar todo el código para readaptarlo a este requerimiento, con todo lo que esto implica.

Decía entonces, que hasta ahora teníamos costos (comisiones y derechos de mercado) que podríamos asumir como "variables puros" en una simplificación.

Pero ¿Qué pasa verdaderamente con el spread?

Para exemplificar lo que quiero decir veamos un ejemplo real de este preciso momento, vamos a comparar spreads en dos criptomonedas, empecemos con el par BTC/USD (Esto es en FTX, uno de los exchanges más líquidos)

Les pego el mismo graf pero duplicado para comparar que pasa si quisiera operar 0.05 BTC o si quisiera operar 1 BTC de capital por trade.

Orderbook				Orderbook			
Grouping: None				Grouping: None			
↓ 47.431				↓ 47.431			
Bid Size (BTC)	Bid Price (USD)	Ask Price (USD)	Ask Size (BTC)	Bid Size (BTC)	Bid Price (USD)	Ask Price (USD)	Ask Size (BTC)
0.0011	47.430	47.431	0.8640	0.0011	47.430	47.431	0.8640
0.0013	47.424	47.432	0.0595	0.0013	47.424	47.432	0.0595
0.0001	47.421	47.435	0.6283	0.0001	47.421	47.435	0.6293
0.0452	47.420	47.439	0.0088	0.0452	47.420	47.439	0.0088
0.0009	47.418	47.444	1.6078	0.0009	47.418	47.444	1.6078
0.3464	47.415	47.445	2.0390	0.3464	47.415	47.445	2.0390
0.6169	47.414	47.449	0.0300	0.6169	47.414	47.449	0.0300
0.6900	47.413	47.457	0.0066	0.6900	47.413	47.457	0.0066
2.8201	47.412	47.460	0.0001	2.8201	47.412	47.460	0.0001
0.9925	47.411	47.465	1.2721	0.9925	47.411	47.465	1.2721
0.0022	47.410	47.466	0.4550	0.0022	47.410	47.466	0.4550

En el primer caso, si quisiera operar solo 0.05 BTC, tengo un spread 47.420 (bid) - 47.431 (ask)

Es decir un spread de 11 USD, en un punto medio de 47.425,5, es decir 0.023% de spread

En cambio, si quisiera operar 1 BTC por trade, tengo el spread: 47.414 (bid) - 47.435 (ask). Es decir un spread de 21 USD, en un punto medio de 47.425,5, es decir 0.044% de spread. O sea, casi el doble de spread que en el caso anterior.

Pero veámos que pasa con un par menos líquido, ejemplo par WAVES / USD, una criptomoneda top100 nativa de su propia blockchain, es decir tampoco estoy agarrando una shitcoin, e insisto en uno de los exchanges más líquidos del mundo, pero vean esta diferencia:

Supongamos que quisiera operar montos similares del orden de los USD 2500 o del orden de los USD 50.000, que serían unos 150 WAVES vs 3.000 (redondeando)

Orderbook				Orderbook			
Grouping: None				Grouping: None			
↓ 16,9710				↓ 16,9710			
Bid Size (WAVES)	Bid Price (USD)	Ask Price (USD)	Avg Size (WAVES)	Bid Size (WAVES)	Bid Price (USD)	Ask Price (USD)	Avg Size (WAVES)
117.5	16,9625	17,0710	150.5	117.5	16,9625	17,0710	150.5
235.0	16,9620	17,0915	650.0	235.0	16,9620	17,0915	650.0
132.5	16,9605	17,1095	150.0	132.5	16,9605	17,1095	150.0
183.0	16,9370	17,1120	152.5	183.0	16,9370	17,1120	152.5
0.5	16,9205	17,1275	1026.5	0.5	16,9205	17,1275	1026.5
19.5	16,8470	17,1305	2198.0	19.5	16,8470	17,1305	2198.0
147.5	16,8465	17,1965	1861.5	147.5	16,8465	17,1965	1861.5
74.0	16,7225	17,3020	3.0	74.0	16,7225	17,3020	3.0
48.0	16,6555	17,3345	2001.5	48.0	16,6555	17,3345	2001.5
74.0	16,6465	17,3670	2994.5	74.0	16,6465	17,3670	2994.5
1.5	16,6160	17,4615	3145.0	1.5	16,6160	17,4615	3145.0

Vean que diferencia, en este par si quisiera operar la cantidad de USD 2500, tengo 16.96 para la compra y 17.07 para la venta, o sea un spread de 0,11 USD para un precio medio de USD 17.02, es decir un spread del 0,64%

Pero si quisiera operar USD 50.000, tengo: USD 16.61 vs 17.13, es decir un spread de USD 0.52 que me representa en el punto medio de USD 16.87 un terrible spread del 3.08% y además tenemos que ver que la punta de compra ni siquiera llega a sumar los USD 50.000 (de todos modos si el Exchange tiene buenos market makers, puedo asumir que si entro a agredir esa punta de compras en forma fraccionada, los bots empezaran a darme, pero entramos en el terreno de la suposición ya)

Como ven, este tema de los spreads, si que es complejo de estimar, Por esto es que lo pongo como semi-variable, es decir que no puedo asumir este costo directamente como un % sobre el monto operado, o mejor dicho:

"Como tengo que asumir el costo de spread como costo transaccional y para simplificar voy a usar un % como si fuera un costo variable, voy a tener que exagerar un poco ese % o calcularlo con mucha data empírica para que luego en producción no me tenga que llevar sorpresas respecto al backtest"

Es la típica solución, tomar el caso mas extremo y el medio y poner un valor mucho mas cercano al caso mas extremo que al medio.

Dicho esto, es importante juntar en un solo costo transaccional variable estos 3 costos, porque de un modo u otro están relacionados, es decir, los exchanges mas chicos, suelen atraer a clientes con mejores comisiones pero tienen mayores spreads y sobre todo para grandes volúmenes, mientras que exchanges mas grandes y consolidados tienen comisiones mas altas, pero mucha mas liquidez para grandes volúmenes y menores spreads. Es un tema a tener en cuenta, hay que armarse una tabla que para el volumen que uno opera compare los diferentes exchanges y mercados para operar y ver que le conviene mas o no, si bien son instrumentos totalmente diferentes, lo pongo de referencia para que me entiendan la idea, vean el siguiente cuadro:

Instrumento	Comisión	Derechos de Mercado	Spread	Costo Total
Acciones Argentinas	0.36%	0.20%	0.10%	0.66%
Acciones USA	0.08%	0.0005%	0.03%	0.11%
Bitcoin	0.13%	0%	0.03%	0.16%

Es una estimación por trade (compra + venta) tomando el costo de los 3 principales exchanges en cada mercado por volumen operado, para una escala de operación de USD 10.000 por trade, para un bot que opere 10 trades por día. Aprovechando las promos y teniendo en cuenta no-residencia en EEUU. En el caso de las acciones EEUU tomé sobre el precio spot de las del SP500 en promedio (porque el costo final de comisión en términos % depende del spot que se tradea).

Hasta acá los costos principales, obviamente a estos tres costos básicos le tendremos que agregar los que siguen, pero al menos estos deberían tenerse en cuenta

Apartado I: Operar sin un mercado real

Antes de seguir con el tema costos transaccionales no puedo dejar de mencionar este tipo de cuestiones. Muchos conocerán plataformas extranjeras que ofrecen 0% de comisión y esas cosas y una facilidad de fondeo con tarjeta de crédito al instante y cosas por el estilo, generalmente acompañadas de atractivas plataformas para operar tanto por PC de escritorio como por los smartphones.

La inmensa mayoría de estas plataformas, escudadas en la regulación de los CFDs o contratos por diferencias, arman esquemas en los que les hacen creer a los minoristas que operan contra un mercado real cuando en realidad los precios mostrados en estas plataformas son precios propios del mismo Exchange o bróker que hace de contraparte la mayoría de las veces, el negocio suele ser el spread, así que ojo ahí porque además los libros de órdenes que se ven no son reales, tienen mucha "operatoria fake" es decir vos ves puntas y supuestas operaciones en un libro de órdenes que pinta

super líquido, pero resulta que si pones una orden real no se opera nunca en forma pasiva, sino que hay que agredir a la punta opuesta (que se va corriendo) para operarse.

Menciono esto acá porque me han consultado algunos por este tipo de plataformas que incluso ofrecen APIs para trading algorítmico algunas, pero les hago esta advertencia porque no es operar en un mercado real, es como una simulación de mercado en donde las puntas las pone el mismo bróker (mas los minoristas que estén operando, o a vece ni eso, les aparecen puntas diferentes a distintos usuarios)

Estos párrafos solo los dejo a modo didáctico, cada uno sabe donde opera y por qué, solo que quería dejar en claro que no siempre la comisión 0% es real. Ojo que si hay casos en los que es real pero no pasa esto en la mayoría. En los casos que crean que es genuino el tema, verifiquen que las puntas realmente sean las mismas que en el mercado real, prueben con muy poco dinero y chequen que se opere realmente etc.

A grandes rasgos lo que habría que preguntarse es ¿Cuál es el negocio si la comisión es 0%? Hay muchas respuestas totalmente válidas, como por ejemplo atraer clientes para vender otro tipo de "servicio vip" o cuestiones similares. Pero tengan cuidado con el tema de las puntas reales y con lo que les voy a comentar en el siguiente apartado acerca del "front-running"

Apartado II: Front Running

Por último, antes de seguir, veamos de que se trata el famoso "Front-Running", quizá muchos de ustedes no lo hayan escuchado y me parece el lugar para dejar esta advertencia por acá.

Muchos casos de plataformas de 0% comisión (el último que me mostraron es uno llamado quantfury o algo así), tienen puntas genuinas que van a parar al libro de órdenes de algún Exchange conocido, es decir que vos pones la orden en la plataforma del "Exchange A" y se ejecuta en el libro de órdenes del "Exchange B", este Exchange B es generalmente super conocido y tiene altísima liquidez.

En principio sería como una intermediación, pero sin comisión o con comisiones incluso mas bajas que las del "Exchange B" que es donde se ejecutan las órdenes. Esto solo ya debería dejar alguna sospecha, chusmeando en un AMA de este tipo de plataformas les consultaron como lograban bajar la comisión del Exchange donde en realidad se ejecutan, y les respondieron como una especie de "mayoreo", es decir que el argumento es que supuestamente este "Exchange B" mas grande les hacía descuento por cantidad. Si bien puede decirse que tiene sentido, es como una manera parcial y bonita de contarlo.

El punto, contado como corresponde, es que este Exchange grande "le compra" el tráfico de órdenes al Exchange chico y las ejecuta luego en su propio Exchange. Es una diferencia sutil, pero notoria

¿Por qué? Porque hay un delay de tiempo entre las órdenes que se ejecutan en el Exchange grande directamente y las que se ejecutan con esta intermediación. Este delay de tiempo a los usuarios les es imperceptible (hablamos de milisegundos) pero lo que hace el Exchange grande con ese flujo de órdenes es lo que llamamos "front-running" es decir adelantarse a las órdenes

Por ejemplo si en BTC contra dólar, en el milisegundo 1 aparece un flujo positivo comprador muy grande en el valor 47.000, lo que hace el Exchange grande es ponerse con un volumen similar en 47.000, pero se pone primero él, cuestión que si se ejecuta esa orden, se ejecuta primero la del Exchange grande e instantáneamente "se da vuelta" o espera que el flujo sea inverso y hace lo mismo..

Estadísticamente hablando el 50% de las veces el flujo será positivo comprador y el otro 50% positivo vendedor, de modo que el Exchange grande al tener esa ventaja de tiempo para colocar las órdenes y poder tapar las órdenes reales antes de que estas lleguen a su libro, siempre sale ganando. El minorista que opera una sola vez, tiene solo el 50% de chances de "ser tapado", si está del lado de la mayoría ponderada por volumen. De modo tal que a la larga el minorista será víctima del front-running, es como que "te lean las cartas" en el truco, porque una cosa es que lean tu orden una vez puesta en el mercado y se apuren a taparte, con lo cual es una carrera de velocidad, y otra muy diferente, es que se adelanten y pongan su orden antes que la tuya porque leyeron la tuya antes de que entre realmente al mercado.

Insisto, que puede haber casos honestos de comisión 0% que apuntan a hacer el negocio vendiendo otros productos a la inmensa cantidad de usuarios que atraen con esa comisión 0%, pero siempre estén atentos porque no es el caso de la inmensa mayoría de este tipo de Exchange.

Impuestos

Como se imaginarán no me voy a detener aquí a detallar costos fiscales, además cada país y cada instrumento es un mundo aparte, pero lo que si voy a mencionar es que, ojo, porque no son costos despreciables, incluso para bots de arbitrajes de tasas y ese tipo de cosas, una cosa es hacerlo con monedas y otra con montos altos en donde hay que considerar cada costo fiscal al detalle.

Muchas veces los bots de arbitrajes de tasas son maravillosos y arrojan resultados muy interesantes, pero cuando se calculan los costos fiscales literalmente deja negativo al trade (respecto a cualquier inversión pasiva con algún beneficio fiscal, generalmente letras soberanas por ejemplo).

Esto suele pasar porque obviamente este tipo de bot tiene un margen de arbitraje muy pequeño porcentualmente a la tasa de referencia, porcentaje que al unir varias operaciones, probablemente alguna de ellas con una fuerte carga fiscal, se evapora rápidamente.

Costos de datos/conexiones

Otro costo que habría que considerar es un prorratoeo en el volumen operado de todos los servicios de datos en tiempo real, APIs, etc que tengamos que pagar para operar.

Obviamente que cuando operamos volúmenes de dinero alto, estos costos son totalmente despreciables, incluso pagando plataformas carísimas como BBG o RTRS, uno dice USD 10k por mes un dineral, pero si operas miles de millones el % es 0.0000 varios ceros y algo

Pero lo marco porque para un minorista a veces USD 100 de acá mas USD 200 de allá empiezan a sumar un valor que prorratoeado en un volumen chico puede ser un % para nada despreciable que tenemos que tomar como un costo a superar solo para salir hechos, es decir que si gasto USD 300 para operar en todo el mes USD 300.000 (entre todas las operaciones sumadas), tengo un costo del 0.1% en cada trade que lo tengo que descontar del resultado de cada trade.

Costos de Infraestructura

De la misma manera que tenemos costos de datos, también hay costos de infraestructura, como un servidor donde vamos a montar el BOT, el servicio cloud que paguemos, el software que tengamos que instalar en el servidor, el servicio de mantenimiento del mismo (aunque lo hagamos nosotros mismos, es un costo que hay que ponderar, no deja de ser un negocio esto y tiene que ser rentable)

Acá hago la misma aclaración que antes, si movemos mucho dinero, este costo es despreciable, pero con poco volumen, puede ser importante, ojo no hablo de "capital" sino de "volumen"

No es para nada lo mismo, pero para que se entienda voy a poner dos ejemplos:

- **CASO 1-** Capital de USD 1.000.000 (muy alto para el 99.9% de los minoristas), que usa un bot de swing trading semanal que rota portafolio cada 23.5 días promedio, rotando el 61% de promedio, de las posiciones en cada operatoria. Este operador tiene un costo de data fee de USD 100 y un costo de infraestructura para montar su bot de USD 100 mensuales.
- **CASO 2-** Capital de USD 10.000 (superado por mas del 90% de los minoristas), que usa un bot de scalping que opera 12.8 veces promedio por día. Tiene los mismos mensuales igual que el CASO 1.

Pregunta:

¿Qué porcentaje de costos de datos + infraestructura deben restar cada uno a cada trade?

A priori ¿parece más complicado el CASO 2 no? (parece que en 1 millón de USD se licua rápido un costo de unos pocos USD mensuales)

Pues verifiquemos haciendo las cuentas

CASO 1

```
capital = 1_000_000
rotacion_por_trade = 0.61
dias_rotacion = 23.5
costo_mensual = 100 + 100
volumen_mensual = capital * rotacion_por_trade * 30/dias_rotacion
costo_por_trade = costo_mensual / volumen_mensual
print(f'El CASO 1 tiene un costo por trade del {costo_por_trade:.4%}')
El CASO 1 tiene un costo por trade del 0.0257%
```

Y para el CASO 2:

CASO 2

```
capital = 10_000
rotacion_por_trade = 1
dias_rotacion = 1/10
costo_mensual = 100 + 100
volumen_mensual = capital * rotacion_por_trade * 30/dias_rotacion
costo_por_trade = costo_mensual / volumen_mensual
print(f'El CASO 2 tiene un costo por trade del {costo_por_trade:.4%}')
El CASO 2 tiene un costo por trade del 0.0067%
```

Como ven, ese mismo costo para el caso 1, aun con 1 millón de dólares de capital representa un costo casi 4 veces mas grande que para el de capital USD 10.000, solo por el tema de la rotación del capital y el volumen operado. Se que parece un tema "pavote" para estos tomos de finanzas quant, pero realmente vi casos reales que quedaron sorprendidos como les afectaba este tipo de costos, aun con grandes capitales porque lo subestimaban, y en la otra punta, minoristas de muy bajo capital que creían muy onerosos ciertos servicios y se sorprendieron fuerte cuando prorratearon en el volumen operado con ese capital.

Suposición de Volumen operado

Un tema final para redondear cuestiones de realismo a agregar en los backtests, es el tema de las suposiciones de volumen operado, muchas veces se asume que se opera un determinado volumen y en la práctica los porcentajes de rendimiento esta ok pero los volúmenes operados son mucho menores. Esto pasa sobre todo, en los bots que no agreden a las puntas, es decir los bots que se colocan a esperar que se opere al menos un pedido por trade. Se da mucho en todo tipo de arbitrajes, pero también en bots que dependen de la microestructura del mercado, para esto es necesario incluir en el backtest un % realista de operaciones esperadas a ejecutarse del total de modo aleatorio.

Muchas veces se operarán las favorables y otras las desfavorables en el backtest, pero es necesario incluir un % en las simulaciones que tome aleatoriamente unas u otras para que el resultado sea mas realista al final de cuentas.

Y antes de cerrar este pequeño título quiero aclarar que una suposición realista en bots de market making debería considerar que va a ser mayor la cantidad de ejecuciones desfavorables que el % de las ejecuciones favorables, es muy complejo de explicar el "por qué" de esto, pero créanme que es así, he backtesteado muchísimo varios bots de market making y colocar 33% vs 67% de ejecuciones a las favorables y desfavorables respectivamente me ha dado muy buenos resultados en cuanto a que coinciden muy bien el backtest con el bot en producción



Otros tipos de Backtest

Hay miles de manera de probar una idea, pero hasta ahora vimos una forma de hacerlo que creemos determinística, es decir, dado un racional de trading, o sea, dados los features y las reglas o gatillos bien claros de compra y venta, ver que trades hubiera ejecutado asumiendo eventos discretos.

¿Qué fue eso de asumiendo eventos discretos?

Claro, si yo considero velas de, supongamos 5 minutos, tengo 4 valores OHLC, correspondientes al momento inicial de la vela, al máximo, al mínimo y al último, es decir tengo 4 instantes fotografiados en la vela, pero no tengo ningún instante intermedio

Pero al menos tengo los valores de todos los instantes intermedios "acotados" entre L y H, es decir se que van a estar "por ahí" entre ambos valores pero no se exactamente donde.

Esto es un problema. Claramente el tiempo es un continuo y el mercado también, no puedo discretizar eventos así nomás, con lo cual todo lo visto hasta acá es simplista en ese sentido, pero no por eso deja de ser una herramienta útil.

¿Cuál es el problema?

Que por ejemplo al cierre de una vela yo puedo "no tener señal de compra" pero en realidad si partía la vela en la mitad del tiempo, quizá justo en esa mitad estaba más cerca del high y "sí me daba señal de compra"

Es decir, yo estoy calculando mi "paso a paso" a los cierres de cada vela (o a las aperturas, o da igual como lo haga, siempre voy a estar haciéndolo a un momento dado, comiéndome todo el resto de los momentos.

Esto hace que el backtest que hagamos no sea tan determinístico como creemos, es decir que si hubiéramos realmente puesto el bot ejecutando esa idea en producción, los trades realizados hubieran sido diferentes, y no solo por esta suposición del tiempo continuo sino por varias suposiciones mas, veamos en detalle cuales:

- Asumimos tiempo discreto
- Asumimos ejecución 100% en un precio de dado (ejemplo el de cierre)
- Asumimos spreads simétricos en torno al precio dado en los momentos discretos utilizados

Veamos uno por uno estos supuestos, y por que están mal y cómo podrían afectarnos y finalmente como atacar este problema y solucionarlo

Asumir el tiempo discreto

Como ya dijimos, cuando tenemos un OHLC estamos asumiendo 4 instantes de tiempo totales en el tiempo que dure esa vela

Veamoslo con un ejemplo, para variar un poco, bajemos datos de ETH desde Binance
Para ello usamos una función sencilla para conectar a su API pública:

```
def historico(symbol, interval='1d', startTime=None, endTime=None, limit=1000):  
    url = 'https://api.binance.com/api/v3/klines'  
    params = {'symbol':symbol, 'interval':interval,  
              'startTime':startTime, 'endTime':endTime, 'limit':limit}  
  
    r = requests.get(url, params=params)  
    js = r.json()  
  
    # Armo el dataframe  
    cols = ['openTime', 'Open', 'High', 'Low', 'Close', 'Volume', 'cTime',  
            'qVolume', 'trades', 'takerBase', 'takerQuote', 'Ignore']  
  
    df = pd.DataFrame(js, columns=cols)  
  
    #Convierto los valores strings a numeros  
    df = df.apply(pd.to_numeric)  
  
    # Le mando indice de timestamp  
    df.index = pd.to_datetime(df.openTime, unit='ms')  
  
    # Elimino columnas que no quiero  
    df = df.drop(['openTime', 'cTime', 'takerBase', 'takerQuote', 'Ignore'], axis=1)  
  
    return df
```

Obtengamos por ejemplo alguna serie ahora usando nuestra función:

```
historico('ETHUSDT', interval='2h').head()
```

openTime	Open	High	Low	Close	Volume	qVolume	trades
2021-09-12 22:00:00	3336.73	3448.00	3331.99	3404.21	36234.2937	1.229481e+08	136870
2021-09-13 00:00:00	3404.30	3430.30	3317.94	3328.51	38888.1575	1.310626e+08	143190
2021-09-13 02:00:00	3328.50	3345.28	3252.45	3257.70	52187.6050	1.719784e+08	137487
2021-09-13 04:00:00	3257.70	3296.82	3240.00	3280.12	46233.6384	1.510961e+08	28482
2021-09-13 06:00:00	3280.11	3288.09	3175.62	3203.57	61821.7155	1.996251e+08	105210

Analicemos lo que pasó el 12 de septiembre entre las 22hs y las 0hs del día siguiente

Si tomamos el precio de cierre, fue 3404.21, quizá si hubiese cerrado a 3420, hubiéramos tenido una señal de compra ahí (en un método de trading imaginario), pero al haber cerrado en 3404.21 no dio ninguna señal. Bien, nada nuevo.. Pero el problema es que en ese lapso de 2 horas hubo ni más ni menos que 136.870 trades, y seguramente muchos de ellos fueron en 3420 o arriba de ese valor

Pero hasta ahí me podrían decir

"No juan, no pasa nada porque yo opero al cierre de cada vela, mi bot espera al cierre de cada vela para meter la operación, por ende si al cierre estaba en 3404.21 y la señal se disparaba arriba de 3420, entonces no se hubiera operado"

¿Seguro no se hubiera operado?

Ahí te puedo decir yo algo así

"Ok, entonces ¿tu método operaría justo a las 0:00 hs del día 13 de septiembre en caso de tener señal de compra, en ese momento exacto?"

Probablemente me responderían algo así:

"Bueno no, tiene una ventana de XX segundos por vela antes del cierre para operar"

Es sensato contar con una ventana de tiempo, para:

- Leer la data feed
- Hacer los cálculos que tenga que hacerse
- Enviar la orden al mercado
- Verificar que se haya ejecutado
- Si no se ejecutó ver por qué no se ejecutó, si se corrió la punta re-evaluar si sigue conviniendo entrar al nuevo precio
- Una vez tomada la decisión cancelar la vieja orden y enviar una nueva
- Volver a chequear si se ejecutó esta vez.. y así hasta asegurarse que está ejecutada la orden

Así que si hubieran respondido que solo se opera al exacto momento del cierre, estamos en problemas, ya veremos en el tomo de implementación de bots que necesitamos una ventana de

tiempo para confirmar órdenes, ni hablar si se tratará de órdenes no-agresivas, es decir, órdenes que crean mercado, o sea que se paran dentro del spread.

Volviendo al diálogo imaginario, habíamos quedado en que el bot necesitaba una ventana de tiempo para operar la orden, entonces volvamos a nuestros datos:

openTime	Open	High	Low	Close	Volume	qVolume	trades
2021-09-12 22:00:00	3336.73	3448.00	3331.99	3404.21	36234.2937	1.229481e+08	136870
2021-09-13 00:00:00	3404.30	3430.30	3317.94	3328.51	38888.1575	1.310626e+08	143190
2021-09-13 02:00:00	3328.50	3345.28	3252.45	3257.70	52187.6050	1.719784e+08	137467
2021-09-13 04:00:00	3257.70	3296.82	3240.00	3280.12	46233.6384	1.510961e+08	86482
2021-09-13 06:00:00	3280.11	3288.09	3175.62	3203.57	61821.7155	1.996251e+08	105210

Supongamos que el bot operaba 30 segundos antes de que termine la vela de las 22hs, y que esperaba un precio de al menos 3420 para comprar

¿Siguen estando seguros de que no hubiera comprado?

El tema es que el HIGH marcó 3448, pero no tengo información si ese HIGH se dio a las 23hs o si se dio a las 23:59:40, es medio exagerado, pero no hace falta que el HIGH se haya dado en ese intervalo de pocos segundos, en realidad cualquier valor entre el cierre y este HIGH.

¿Parece poco probable no?

Bueno a ver, hagamos cuentas, dividamos las 2 horas en 7200 segundos, imaginemos que de los 136.870 trades ejecutados se ejecutan parejos en el transcurso de esas 2 horas, tenemos que en 30 segundos que es nuestra ventana, hubo 570 trades (nada poco ehh)

Ahora ¿que probabilidad tenemos que haya habido justo uno arriba de 3420?

Imposible saberlo con solo estos datos, pero podemos asumir cosas y estimar, por ejemplo, si del rango min-max hubiese estado todos los trades distribuidos perfectamente, teníamos un rango de 116.01 USD en los 7200 segundos, y el rango que disparaba la orden era arriba de 3420, con lo cual arriba de eso y hasta los 3448 que fue el max, tenemos 28 USD, es decir:

$$28 / 116.01 = 24\% \text{ del recorrido de la vela}$$

Dijimos que en nuestro intervalo teníamos 570 trades de 136.870 ¿Qué probabilidad hay que entre esos 570 aparezca uno solo perteneciente a ese 24%?

Ya se, ya sé imposible saberlo, además ni los trades se distribuyen uniformemente entre el min y max, ni se dan en forma absolutamente aleatoria entre los 7200 segundos

Conclusión, es imposible saber si el gatillo se hubiera disparado en esos últimos 30 segundos de vela

O sea que ¿todo lo que hicimos hasta ahora en el framework de backtesting que explicamos es absolutamente al pedo, porque no podemos asumir operar al cierre?

Si pero no

Jaja, no se enojen, si les decía eso de entrada nadie en su sano juicio hubiera leído las +200 páginas siguientes. Ahora el punto es que en realidad si bien lo que creímos determinístico no lo es tanto, tampoco nos cambia tanto la vida, ¿Por qué?

Sencillamente una parte de los trades que se ejecutaban en la vela "N" en nuestro backtest, se hubieran ejecutado en la vela "N-1" en la vida real y otra parte en la vela "N"

Ahora el efecto es neutro, porque algunos de los trades que se ejecutaban en realidad en "N-1" (o podría ser "N-2" u otra vela..) decía algunos de esos trades se hubiera ejecutado levemente por encima del precio de cierre de la vela "N" y otra parte levemente por debajo. Lo he chequeado con bots que han operado cientos de miles de trades y el efecto termina anulándose. Con lo cual la conclusión es que si bien el backtest no es tan determinístico como parece inicialmente, en la práctica las diferencias no afectan significativamente.

Asumimos ejecución 100% en un precio de dado

Seguimos con el siguiente problemita en nuestro framework planteado. Si prestan atención estamos asumiendo (sin decirlo) que todo trade hubiese sido ejecutado.

Pero esta simple afirmación que parece trivial, porque uno dice si la punta de venta dice 10 papeles x 100 USD cada uno, y yo tengo que comprar a USD 100, mando la orden y listo.

El tema es que a veces pasa que mandas la orden y cuando llega, esos 10 papeles a USD 100 ya no están ahí. No es lo mas común en trades agresivos con las puntas, pero cada tanto pasa.

Y acá quiero hacer una pequeña aclaración, en este caso el efecto no es neutro, es direccional, es decir, cuando justo el papel en cuestión "se vuela" en esa vela, y nosotros queríamos comprar, lo mas probable que es el verdadero precio a pagar en la vela "N+1" donde el bot real seguirá dando gatillo de compra seguramente, sea bastante más alto. Si embargo en el caso que el papel se caiga, vamos a ejecutar nuestra orden de compra en la vela "N" a un precio más abajo favoreciéndonos, pero no con tanta diferencia para compensar los que son en contra porque estamos comparando un deslizamiento pequeño de unos segundos con una vela entera.

¿Forma de solucionar este inconveniente para darle mas realismo a un backtest?

Yo no encontré, quiero decir, lo único que se me ocurre es probar con un bot real, que meta muchas operaciones y, luego realizar un backtest para atrás (teniendo en la mano ya los datos del bot real)

Y con esto tener un % para aplicarle montecarlo selectivo a los backtest, ahora explico esto

Antes que nada decir también que ese % es muy relativo a cada tipo de trade, timeframe, mercado, activo, tendencia, etc, generalmente ese % de ejecución es muy alto, pero nunca el 100%, en mi experiencia el % de ejecución (en la vela "N") del bot real vs lo que marcó el backtest posterior fue del rango del 93% al 99% redondeando, pero insisto, timeframes cortos, mercado cripto, volatilidades altas, mercado continuo 24x7, es decir, quizás el mismo tipo de operativa en otro mercado marque % de ejecución diferente. Yo creo que no hay manera de saber esto hasta no probarlo en producción (además otra cosa que no dije es que también esos % me variaron bastante con la calibración de parámetros)

De todo modo, asumiendo un 95% vamos a poner que es un % creo que razonable de ejecución, lo que podemos hacer para backtestear es poner un filtro aleatorio que tome solo el 95% de los trades en la vela "N" y el resto intente ejecutarlos en la vela "N+1" y así siempre y cuando siga la condición de gatillo. Esto es bastante más realista pero complejiza el backtest y el resultado no varía tanto, ustedes evaluarán cuando conviene y cuando no el esfuerzo. Yo en el libro no quise complicarla demasiado pero lo cierto es que cuando uno ya tiene cierta experiencia y varios bots en la cancha, empieza a armarse clases y una orientación a objetos de su código y esto simplifica la introducción de varios de estos agregados, obviamente al principio armar la arquitectura de esas clases es medio un chino con muchos idas y vueltas pero a la larga vale la pena, el tema es que es para un nivel más avanzado.

Asumimos spreads simétricos en torno al precio operado

Este es otro problema, por ejemplo si yo mido que el spread medio es del 0.2% por decir algo, si tengo que operar algo cuyo último valor operado fue de USD 100, puedo asumir que el bid estará en \$ 99.9 y el ask en \$ 100.1, de este modo tengo una brecha de \$ 0.2 que representa el 0.2% de \$100

Pero ¿de donde saco que si el último trade fue en \$100, voy a tener ese bid y ese ask?

Miren esto:

Orderbook				Market Trades		
Bid Size (BTC)	Bid Price (USD)	Ask Price (USD)	Ask Size (BTC)	Price (USD)	Size (BTC)	Time
↓ 49.492						
0.7943	49.491	49.492	2.1754	49.495	0.0229	4:43:28
0.4257	49.488	49.493	0.0087	49.497	0.0010	4:43:28
0.4678	49.487	49.494	0.0595	49.498	0.0058	4:43:28
0.0003	49.486	49.496	0.6045	49.500	0.0500	4:43:27
0.0013	49.484	49.499	0.4170			
0.0009	49.483	49.500	0.0372			
1.7919	49.482	49.501	1.1543			
0.0001	49.481	49.502	3.4316			
0.0004	49.478	49.503	0.0472			
3.5095	49.477	49.507	0.0413			
5.5352	49.476	49.509	1.6580			

Como pueden ver en la página anterior, capturando un instante cualquiera de un libro de órdenes de BTC, veo que tenemos un spread en 49.491 y 49.492 mientras el último operado fue 49.495 (afuera del spread). Los valores del libro resaltado son porque en ese mismo instante fueron modificadas las cantidades de las órdenes a esos precios.

En el caso de esta instantánea se ve claramente como los vendedores agredieron a los compradores, ya que 4:43.27 se operó a 49.500 y un segundo mas tarde salieron 3 operaciones a 49.498, 49.497 y 49.495 respectivamente todas en el mismo segundo, y en ese mismo instante el próximo vendedor tendrá que conformarse con 49.491 que es la punta compradora.

A lo que voy, si en ese preciso instante cerrara la vela, yo asumiría que voy a operar al último precio operado (49.495) + el spread, pero la realidad es que si coloco la orden de compra operaría a 49.492 que es mas bajo. Ojo también puede pasar para el otro lado que se deslice el spread para arriba en lugar de para abajo como pasó en este caso.

Afortunadamente, este efecto es neutro, es decir, podemos asumir simetría de la caja de puntas vs el último precio de cierre, lo he probado con infinidad de activos y mercados y en "la foto" instantánea es siempre neutro, es decir se compensa, el promedio del último precio operado cae siempre en medio del spread (en promedio con millones de instantáneas)

Bueno, todos estos pequeños detalles del libro de órdenes, me llevan al siguiente nivel que es armar un backtest con sucesos tick by tick.

Sucesos con Tick by Tick

Aclaración, no desarrollé de entrada este tema en este tomo porque encararlo bien, requiere un buen uso de bases de datos, gran cantidad de data y demás cuestiones que van a ser abordadas en el tomo siguiente, sin embargo, si bien no vamos a plantear un backtest completo con este tipo de sucesos, voy a mostrarles como obtener este tipo de data, que tipo de data es, y como incide esto en un backtest de la misma forma que lo hicimos con velas de intervalos de tiempo discretos.

O sea, voy a plantear el tema para, en el tomo siguiente, poder explorar mejor como manejar estas grandes cantidades de datos, luego ustedes podrán unir cabos sueltos y armar lo mismo que hicimos con las velas, pero con ticks y puntas del order book apuntando a una base de datos previamente construida.

Así que arranquemos, la idea del backtest "tick by tick" consiste en tomar como data feed, en lugar de velas que representan los precios en un determinado rango, tomar cada instante en el que se haya

producido un trade, es decir ya no vamos a tener OHLC, sino que cada instante tendrá su único precio operado.

Como se podrán imaginar es muy diferente, porque habrá segundos que tengan 100 trades, y habrá segundos que tengan 10 o ningún trade es decir que vamos a tomar cada operación como un instante único en el tiempo.

Dirán: ¿Podría tomar todas las de cada segundo y juntarlas en una especie de promedio?

No, porque sería el equivalente a armar velas de 1 segundo, es decir, necesitamos algo que represente lo mas parecido posible a una variable continua en el tiempo, cualquier rejunte de operaciones por intervalos predefinidos es armar un híbrido con todas las desventajas que vimos en el punto anterior que son las que queremos eliminar justamente.

Pero eso no es todo, no alcanza con tener solo los "ticks" o trades individuales

Necesitamos los "eventos" es decir los cambios del libro de órdenes, cada cambio del libro de órdenes es un evento que modifica la situación, por mas que el último trade se haya operado a cierto nivel de precio, que podemos tomar como válido, lo cierto es que lo que definirá el precio del próximo trade es el libro de órdenes y cualquier cambio sobre él.

Entonces necesitamos descargar dos tipos de datos:

- Los trades (uno por uno)
- Los estados del libro de órdenes (en cada instante en que se produce un trade)

Veamos como se ve este tipo de data y como obtenerla

Para ello voy a usar la API de Binance que es pública y me permite traer esto de todas las criptomonedas y sintéticos de acciones también.

Pueden encontrar la documentación de su API en:

- <https://binance-docs.github.io/apidocs/spot/en>

Si bien pueden sacarse un API key para tener acceso a determinados endpoint, como todos los referidos a estado de cuenta, ruteo de operaciones etc, para lo que es data pública por lo general se accede sin API, excepto para "consultas pesadas"

Vamos entonces a crear una función que lea los trades de Binance del par BTC-USD

Y vamos a traer también el libro de ordenes en un instante dado.

Vamos a importar requests, como siempre para conectar a una API, y pandas para facilitar el trabajo y armado de set de datos final.

```
import requests
import pandas as pd
```

Traemos data Tick by Tick

Armamos una función para traer la data:

```
def trades(symbol, limit=1000):
    url = 'https://api.binance.com/api/v3/trades'
    params = {'symbol':symbol, 'limit':limit}
    r = requests.get(url, params=params)
    js = r.json()
    df = pd.DataFrame(js)

    # Convertimos a numericas algunas columnas
    convertir = ['price','qty','quoteQty']
    df.loc[:,convertir] = df.loc[:,convertir].apply(pd.to_numeric)

    #pasamos el timestamp a fecha
    df.time = pd.to_datetime(df.time, unit='ms')
    df.set_index('time', inplace=True)

    return df.round(2)
```

Y traemos la data:

```
tr = trades('BTCUSDT', 100)
tr.tail(10)
```

time	id	price	qty	quoteQty	isBuyerMaker	isBestMatch
2021-12-05 08:29:28.755	1178092421	49338.22	0.00	151.96	True	True
2021-12-05 08:29:29.063	1178092422	49338.23	0.00	222.02	False	True
2021-12-05 08:29:29.071	1178092423	49338.23	0.00	127.29	False	True
2021-12-05 08:29:29.111	1178092424	49338.23	0.05	2311.50	False	True
2021-12-05 08:29:29.369	1178092425	49338.23	0.00	19.24	False	True
2021-12-05 08:29:29.607	1178092426	49338.23	0.02	1036.60	False	True
2021-12-05 08:29:29.662	1178092427	49338.22	0.00	21.71	True	True
2021-12-05 08:29:29.662	1178092428	49338.22	0.01	573.80	True	True
2021-12-05 08:29:30.052	1178092429	49338.22	0.00	125.81	True	True
2021-12-05 08:29:30.279	1178092430	49338.22	0.00	30.10	True	True

Ahí tenemos las últimas 10 operaciones, como verán, a las 8:29:29 segundos, tuvimos 7 operaciones, podemos ver si fue el comprador creador de mercado, las que tienen True ahí significa que fue el vendedor quien lo agredió, es decir que el comprador puso el precio y el vendedor lo tomó. Si dice False lo contrario obviamente.

Y como verán en esas 10 operaciones una diferencia ínfima de 0.01 en un precio de casi 50.000, y tenemos operaciones desde 19 a 2311 USD.

Bien, pero se estarán preguntando ¿y cuantas operaciones serán en una vela de 5 minutos? Bueno ya habíamos visto en otro Exchange que eran miles, así que si, efectivamente la cantidad de data del tick by tick es enorme, comparada con la cantidad de data de las velas, pero enorme.

Y si la cantidad de data de cada trade es enorme, ¿Qué hay del libro de órdenes?

Bueno, para que se vayan haciendo una idea, por cada trade, el libro de órdenes se modifica mas de 10 veces, así que sí, 10 veces mas grande, por lo menos es la info de los eventos en el libro de órdenes.

Traemos data Libro de órdenes

Bueno, traigamos entonces datos del libro de órdenes, un primer intento sería hacer algo así como vimos en el tomo de APIs:

```
def getBook(symbol):
    url = 'https://api.binance.com/api/v3/depth'
    p = {'symbol' : symbol}
    r = requests.get(url, params=p)
    js = r.json()
    bids = pd.DataFrame(js['bids'])
    asks = pd.DataFrame(js['asks'])
    df = pd.concat([bids,asks],axis=1)
    df.columns = ['bid_q','bid_px','ask_px','ask_q']
    df = df.reindex(columns=['bid_q','bid_px','ask_px','ask_q'])
    df = df.apply(pd.to_numeric).round(4)
    return df

getBook('BTCUSDC')
```

	bid_q	bid_px	ask_px	ask_q
0	48948.70	0.1000	0.1000	48959.73
1	48948.63	0.0406	0.0378	48961.27
2	48943.65	0.0511	0.1079	48961.28
3	48937.23	0.0440	0.0480	48978.99
4	48934.30	0.0511	0.1601	48983.72
..
95	48036.85	0.0006	0.0070	49800.00
96	48034.80	0.0206	0.0004	49815.50
97	48020.00	0.0416	0.0017	49831.11
98	48017.61	0.0005	0.0004	49843.15
99	48005.39	0.0924	0.3382	49850.06

100 rows × 4 columns

Esta buenísimo, me trae la cola de 100 posiciones del libro de órdenes, todos los bids y asks con sus precios y cantidades, pero el tema es que esto es solo "una foto". Es decir es el estado del libro de órdenes en un instante dado, pero yo necesito "una película" para poder backtestear, necesito la dinámica de como se va modificando el libro de órdenes por cada trade, es decir que si en el bloque de código del título anterior obtenía en 1 segundo como 10 operaciones de BTc, seguramente en ese mismo lapso haya habido mas de 100 modificaciones la libro de órdenes, o sea gente que va colocando órdenes que no se ejecutan, piensen en su operatoria manual ¿cuántas veces pusieron órdenes que no se ejecutaron?, imaginen que los bots es mucho más.

Entonces se preguntarán ¿Cómo recabo semejante cantidad de información para luego backtestear? Claramente, no voy a mandar 10 requests por segundo consultando el libro de órdenes por cada décima de segundo, primero porque necesito muy buena infraestructura para coordinar esto con hilos e ir guardando la data en tiempo real 24x7 y que no explote nada, y segundo porque tampoco me serviría.

Imaginen que tengo el estado del libro de órdenes en el instante
00 hs 00 minutos 00 segundos y 100 milisegundos

Luego tengo el estado en el instante:
00 hs 00 minutos 00 segundos y 200 milisegundos

Pero tuve dos transacciones en los instantes:
00 hs 00 minutos 00 segundos y 125 milisegundos
00 hs 00 minutos 00 segundos y 379 milisegundos

Lo que me interesaría saber es antes y después de cada una de esas transacciones como estaba el libro de órdenes y si consulto cada 0.1 segundos (100 milisegundos) me pierdo esa info.

La solución es usar websockets, pero hasta ahí, ¿por qué digo hasta ahí?
Bueno, porque dependiendo de la API voy a tener cierta granularidad máxima (mínima) del cambio del libro de órdenes, es decir si el Exchange recibe 10.000 cambios por segundo, no me va a traer cada cambio ocurrido dentro del rango de 1/10.000 segundos, sino que va a agrupar varios cambios y cada tanto tiempo me mandará todo junto, esto es porque se necesita una infraestructura muy potente para abastecer websockets con tanta granularidad

O sea, del mismo modo que en los endpoints tengo un límite en la cantidad de requests por segundo, en los websocket hay un límite de granularidad en los cambios informados.

En el caso del websocket del order book de Binance esta granularidad es de 100ms, es decir 1/10 de segundo. No es muy fina pero es bastante razonable, obviamente en todos los exchanges si operas grandes volúmenes podés acceder generalmente a mejores límites de data feed, o incluso muchos tienen una infraestructura FIX para acceder a un datafeed mas rápido.

Volviendo al tema del websocket para traer cambios en toda la profundidad del order book, lo mejor que podemos lograr con la API pública de Binance sería traer cada 100 milisegundos todos los cambios agrupados en todo el order book en toda su profundidad.

Otra opción que veremos después es para traer en real time, absolutamente todo cambio realizado, no importa si son mucho mas de 10 por segundo, pero solo en las mejores puntas del bid y ask, es decir en las puntas calientes, esto es porque hay mucho **spoofing** en crypto entre market makers. A quienes no sepan que es el "spoofing" los invito a googlear porque voy a desarrollar mucho mejor esto en el tomo de bots de trading donde desarrollaré algo específico de market making.

Bueno, ya me estaba yendo por las ramas, volviendo a lo mejor que puedo lograr en profundidad de todo el libro de órdenes con la API pública de Binance, sería algo así:

Como pequeño repaso por si no practicaron mucho con los tomos de APIs, para traer de websocket tenemos que tener la librería websocket, pero especialmente websocket-client porque la otra no funciona bien para este tipo de ejemplo. Por eso el comentario de lo que hay que instalar desde el anaconda prompt por consola. Luego los imports necesarios como siempre.

```
# pip install websocket-client

# Imports
import json
import websocket
from datetime import datetime
import pandas as pd
```

Y luego el armado de la conexión, esperando el primer mensaje de bienvenida

```
socket='wss://stream.binance.com:9443/ws'
conn = websocket.create_connection(socket)

# Creamos el JSON con los datos de suscripcion
subscribe_message = '{"method": "SUBSCRIBE", "params":["btcusdt@depth@100ms"], "id": 1}'

# Nos suscribimos al websocket
conn.send(subscribe_message)

# Recibimos datos
conn.recv()

'{"result":null,"id":1}'
```

Como ven me suscribo a la profundidad con una granularidad de 100 ms, no intenten poner una menor porque Binance se las rechaza. Pueden poner una de 1000 ms, pero no menos a 100.

Y luego armo un FOR para recibir los primeros 10 mensajes y los meto en un dataframe:

```
conn.recv()

results = []
for i in range(10):
    data = json.loads(conn.recv())
    r = {'EventTimeUnix': data.get('E'), 'EventTime': datetime.fromtimestamp(int(data.get('E'))/1000),
         'bid':data.get('b'), 'ask':data.get('a')}
    results.append(r)

pd.DataFrame(results)
```

	EventTimeUnix	EventTime	bid	ask
0	1638762542765	2021-12-06 00:49:02.765	[49055 92000000, 0 00000000], [49052 72000000, 0 00000000]	[49063 79000000, 1 89363000], [49065 60000000, 0 00000000]
1	1638762542865	2021-12-06 00:49:02.865	[49063 78000000, 0 14779000], [49058 69000000, 0 00000000]	[49063 79000000, 1 95361000], [49071 92000000, 0 00000000]
2	1638762542965	2021-12-06 00:49:02.965	[49063 47000000, 0 15268000], [49058 68000000, 0 00000000]	[49063 79000000, 1 97061000], [49065 70000000, 0 00000000]
3	1638762543065	2021-12-06 00:49:03.065	[47307 69000000, 0 00061000]	[49067 53000000, 0 50000000], [49068 50000000, 0 00000000]
4	1638762543165	2021-12-06 00:49:03.165	[49048 20000000, 0 00000000], [49022 88000000, 0 00000000]	[49067 53000000, 0 00000000], [49068 82000000, 0 00000000]
5	1638762543265	2021-12-06 00:49:03.265	[49042 57000000, 0 00000000], [49042 18000000, 0 00000000]	[49068 50000000, 0 23889000]
6	1638762543365	2021-12-06 00:49:03.365	[49063 78000000, 0 14754000], [49042 53000000, 0 00000000]	[49067 53000000, 0 50000000], [49068 50000000, 0 00000000]
7	1638762543465	2021-12-06 00:49:03.465	[48965 65000000, 0 04683000]	[49065 70000000, 0 00000000], [49067 53000000, 0 00000000]
8	1638762543565	2021-12-06 00:49:03.565	[49063 78000000, 0 15433000], [49063 76000000, 0 00000000]	[49069 76000000, 0 00000000], [49080 49000000, 0 00000000]
9	1638762543665	2021-12-06 00:49:03.665	[49063 78000000, 0 13984000], [49063 65000000, 0 00000000]	[49063 79000000, 1 97061000], [49066 19000000, 0 00000000]

Lo pongo en un DataFrame porque me resulta practico para mostrarlo en el libro, pero bien podrían guardar una lista de diccionarios, vean el último "r" queda así:

```
r
{'EventTimeUnix': 1638762543666,
 'EventTime': datetime.datetime(2021, 12, 6, 0, 49, 3, 666000),
 'bid': [['49063.78000000', '0.13984000'],
 ['49053.65000000', '0.00000000'],
 ['49052.73000000', '0.13959000'],
 ['49052.56000000', '0.00000000'],
 ['49051.36000000', '0.00000000'],
 ['49039.05000000', '0.00000000'],
 ['49038.52000000', '0.00000000'],
 ['49038.51000000', '0.00000000'],
 ['49036.84000000', '0.50000000'],
 ['49015.89000000', '0.00000000'],
 ['48983.44000000', '0.99953000']],
 'ask': [['49063.79000000', '1.97061000'],
 ['49066.19000000', '0.69900000'],
 ['49067.92000000', '0.00000000'],
 ['49067.93000000', '0.00000000'],
 ['49068.50000000', '0.30849000'],
 ['49068.84000000', '0.00000000'],
 ['49068.87000000', '0.50000000'],
 ['49068.88000000', '0.00000000'],
 ['49068.92000000', '0.00000000'],
 ['49071.61000000', '0.00000000'],
 ['49137.97000000', '0.32210000']]}
```

Es lo mismo que figura en la última fila del datafram obviamente, pero en formato diccionario para apreciarlo entero mucho mejor. En fin, con toda esta data tienen para entretenese armando backtests más realistas

Si necesitan absolutamente todos los updates de cada bid y ask al menos de los hot, esos que quedan primeros antes y después de cada trade, lo que pueden hacer es suscribir al socket de las puntas en real time, no tienen la profundidad de todo el libro, pero si tienen el update en tiempo real con la granularidad real sin limitaciones. Les pego lo que informa Binance

The screenshot shows the Binance API documentation for Stream endpoints. The main navigation bar includes links for Binance, Spot/Margin/Savings/Mining, USD-M Futures, COIN-M Futures, and Vanilla Opt.

Individual Symbol Book Ticker Streams

Pushes any update to the best bid or ask's price or quantity in real-time for a specified symbol.

Stream Name: <symbol>@bookTicker

Update Speed: Real-time

Esto es en tiempo real, sin limitación alguna, es decir trae todos los cambios en los mejores bids y asks

All Book Tickers Stream

Pushes any update to the best bid or ask's price or quantity in real-time for all symbols.

Stream Name: !bookTicker

Update Speed: Real-time

Partial Book Depth Streams

Este trae una profundidad de 20 en ambas colas, pero con un límite de granularidad de 100 ms.

Stream Names: <symbol>@depth<levels> OR <symbol>@depth<levels>@100ms

Update Speed: 1000ms or 100ms

Veamos entonces como traer las mejores puntas en tiempo real sin limitación de granularidad
La suscripción al socket sería igual, solo cambian los params:

```
socket='wss://stream.binance.com:9443/ws'
conn = websocket.create_connection(socket)

# creamos el JSON con los datos de suscripción
subscribe_message = '{"method": "SUBSCRIBE", "params": ["btcusdt@bookTicker"], "id": 1}'

# Nos suscribimos al websocket
conn.send(subscribe_message)

# Recibimos datos
conn.recv()

'{"result":null,"id":1}'
```

Y vamos a recibir entonces, por poner algo, 1000 updates, y de paso veamos cuanot tiempo tarde en recibir estos 1000 cambios en BTC en las mejores puntas

Miren esta locura, 83 milisegundos en recibir 1000 cambios en las puntas, es decir en menos de una décima de segundo hubo 1000 cambios en los bids y asks punteros

```
%time
conn.recv()
results = []
for i in range(1000):
    data = json.loads(conn.recv())
    results.append(data)

df = pd.DataFrame(results)
df
```

Wall time: 86.3 ms

	u	s	b	B	a	A
0	15462955028	BTCUSDT	49119.51000000	0.91724000	49119.52000000	0.04357000
1	15462955043	BTCUSDT	49119.51000000	0.91724000	49119.52000000	0.04308000
2	15462955045	BTCUSDT	49119.51000000	0.91425000	49119.52000000	0.04308000
3	15462955048	BTCUSDT	49119.51000000	0.91425000	49121.92000000	0.02110000
4	15462955049	BTCUSDT	49119.51000000	0.89567000	49121.92000000	0.02110000
...
995	15462961099	BTCUSDT	49110.35000000	0.31884000	49110.36000000	0.34430000
996	15462961108	BTCUSDT	49110.35000000	0.31884000	49110.36000000	0.33934000
997	15462961112	BTCUSDT	49110.35000000	0.31884000	49110.36000000	0.35162000
998	15462961119	BTCUSDT	49110.35000000	0.31679000	49110.36000000	0.35162000
999	15462961125	BTCUSDT	49110.35000000	0.31679000	49110.36000000	0.30566000

1000 rows × 6 columns

Vamos a arreglas un poco este dataframe para entenderlo mejor

Le pongo nombres más legibles por humanos a las columnas:

```
df.columns = ['update_id','symbol','bid','bid_q','ask','ask_q']  
df
```

	update_id	symbol	bid	bid_q	ask	ask_q
0	15462955028	BTCUSDT	49119.51000000	0.91724000	49119.52000000	0.04357000
1	15462955043	BTCUSDT	49119.51000000	0.91724000	49119.52000000	0.04308000
2	15462955045	BTCUSDT	49119.51000000	0.91425000	49119.52000000	0.04308000
3	15462955048	BTCUSDT	49119.51000000	0.91425000	49121.92000000	0.02110000
4	15462955049	BTCUSDT	49119.51000000	0.89567000	49121.92000000	0.02110000
...
995	15462961099	BTCUSDT	49110.35000000	0.31884000	49110.36000000	0.34430000
996	15462961108	BTCUSDT	49110.35000000	0.31884000	49110.36000000	0.33934000
997	15462961112	BTCUSDT	49110.35000000	0.31884000	49110.36000000	0.35162000
998	15462961119	BTCUSDT	49110.35000000	0.31679000	49110.36000000	0.35162000
999	15462961125	BTCUSDT	49110.35000000	0.31679000	49110.36000000	0.30566000

1000 rows x 6 columns

Le voy a sacar la columna de símbolo para pasar todo a numérico de una:

```
df.drop(['symbol'], axis=1, inplace=True)  
df
```

Y lo paso a numérico de una todas las columnas, para poder tirar una descriptiva de los campos:

```
df = df.astype(float)  
df.iloc[:,1:].describe()
```

	bid	bid_q	ask	ask_q
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	49127.035810	0.269828	49127.650170	1.006986
std	8.515126	0.369627	8.604833	2.341458
min	49110.350000	0.000010	49110.360000	0.000010
25%	49119.547500	0.049190	49121.110000	0.032600
50%	49127.150000	0.143945	49127.160000	0.181490
75%	49131.530000	0.320207	49131.540000	0.610073
max	49146.280000	2.403620	49146.290000	8.827010

Y vean todo lo que paso en 86 milisegundos con las puntas del BTC,

Empecemos por entender la guerra de market makers:

- El bid osciló entre 49110.35 y 49146.28
- El ask osciló entre 49110.36 y 49146.29

¿Se dan cuenta?

Estos son todos bots de market makers jugando entre el spread pero sin pegarse entre ellos, probablemente alguno pescó algún trade de un humano que colocó alguna orden manual entre las puntas en esos 86 milisegundos.

Otras conclusiones, veamos las cantidades:

- La media de cantidades del bid fue 0.269 BTC +/- 0.36
- La media de cantidades del ask fue 1.006 BTC +/- 2.34
- La máxima cantidad en el bid fue 2.4 BTC
- La máxima cantidad en el ask fue 8.82 BTC

Esto nos muestra que durante esos 86 milisegundos los vendedores estuvieron mas cargados que los compradores, es muy probable (de mantenerse un tiempo esta diferencia) que los market makers vengan mas tomadores, por eso cargan mas las puntas de venta, otro tema mas de bots de market makers que veremos en el tomo de bots, no me quiero colgar con algo que nada que ver con el tema.

En definitiva, cerrando un poco este tema de las lógicas tick by tick en el backtest, como ven, la información de "las velas" es interesante pero poco profunda, ahora si voy a querer probar un bot que mete swings semanales, la verdad me alcanza y sobra con velas diarias para probar mi idea, sin embargo, si quisiera hacer un bot de arbitrajes que tiene que jugar fino con las puntas del bid y el ask, quizá necesite data del order book y de cada tick operado para backtestear.

Dejo abierto para el próximo tomo la dificultad que se nos abre si queremos empezar a mejorar y profesionalizar nuestro framework de backtesting.

Dificultades a resolver (lista no taxativa, solo nombro algunas con las que nos fuimos topando):

- Necesitamos un mejor orden y organización de los datos que vamos bajando, así luego accedemos a ellos en diferentes backtests sin tener que bajar a cada rato data nueva
- Necesitamos por ejemplo tablas con los tickers del SP500 y todos los índices que nos interesen para cada fecha para evitar sesgos como el de supervivencia y selección.
- Necesitamos data mas pulida que lo que nos trae Yahoo Finance, automatizar si es posible el data profiling de los datos que conseguimos
- Necesitamos una infraestructura capaz de dejarla corriendo y que todos los días baje datos actualizados y los guarde en algún lado

Como se imaginarán, para gestionar todo esto y correr los backtest con mejor calidad de data, es necesario ir armándonos nuestra propia BASE DE DATOS, es uno de los temas claves del próximo tomo

Otro tema clave es la parte mas tecnológica de montar un servidor en alguna nube y poder dejar automatizado un proceso que todos los días se actualice solo y actualice nuestra base de datos y todos los procesos de data profiling y demás, cosa que cuando quiera acceder a mi base de datos, esta esté lista ahí en la nube para mi. Esto me lleva a otro tema clave que es el tema de infraestructura cloud.

Y por último, si queremos hacer todo esto en forma ordenada y sin volvemos locos para actualizar cada cosa que queramos agregar en un futuro, deberemos trabajar con algún framework como Flask o Django, que es otro de los temas clave del próximo tomo.

Así que ya saben lo que viene, Django, Servidores cloud, Bases de Datos, y para enseñar todo esto, vamos a darle un sentido práctico y vamos a correr procesos para realizar screeners periódicos y armarnos nuestro propio sistema de alertas personalizado

Bueno, hasta acá llegamos por ahora entonces, nos vemos en el próximo tomo

La colección Python para finanzas Quant

El presente libro es parte de una colección de varios tomos que tiene como objetivo hacer dar los primeros pasos en programación a gente que nunca antes programó, tiene la particularidad de estar enfocado en temas de mercado de capitales ya que es parte de una serie de libros que van desde los primeros scripts hasta terminar en temas mas complejos como el análisis cuantitativo y research de sistemas de inversión, backtestings, screeners, bots de trading entre otros ejemplos.

Me enfoqué en la parte práctica ya que entiendo que la programación es una disciplina que se aprende y se asimila con horas y horas de sentarse a resolver problemas. Esto es 80% práctica y 20% teoría, aprender a programar es como aprender a manejar, la cantidad de horas al teclado es lo que hará la diferencia.

Creo que tanto la programación como las finanzas son dos campos importantes en la formación profesional y absolutamente imprescindibles para la planificación personal laboral y patrimonial en un mundo cada vez mas interesante, complejo y competitivo. Por eso espero que sea de utilidad y sea la puerta de entrada que termine acercando a mucha gente a este mundo tan fascinante y desafiante de la programación y de las finanzas cuantitativas.

Este tomo T[7]

La idea de este tomo es darles un esquema o estructura de pasos a seguir para realizar lo que en finanzas quant llamamos "Backtesting" que tiene como objetivo poner a prueba la idea de trading algorítmico y testearla en diversos escenarios, con diversos parámetros y diversos activos, a ver como se hubiera comportado la estrategia en todo tipo de contexto, para al fin de cuentas, decidir si puede ser una buena idea a aplicar o no.

Hay muchas voces a favor y en contra de los backtest, generalmente los que se oponen o menosprecian su potencial lo ven como la demostración de un teorema, esto no es ciencia exacta, que un backtest arroje resultados alentadores no significa que en el futuro vaya a funcionar igual en el mercado, pero nos da una pauta, una referencia y nos arroja decenas de indicadores y ratios super interesantes para evaluar las estrategias a utilizar, con lo cual la idea de este tomo es, tener muchas herramientas mas para tomar decisiones.