BOTS DE TRADING ALGORÍTMICO

Introducción

El presente documento tiene por objeto introducir una serie de proyectos vinculados con la inversión mediante estrategias de trading algorítmico, con el propósito de abordar la materia de estudio de forma sistemática y eficiente. A su vez, se propone analizar los resultados obtenidos con el fin de evaluar el alcance y la eficacia de las distintas metodologías de inversión que pueden implementarse a través de esta técnica.

A lo largo del desarrollo se hará especial énfasis en aspectos relacionados con el funcionamiento del mercado y en la manera en que dichos elementos fueron analizados para su posterior integración en los proyectos. En este marco, se contemplarán diversas aproximaciones y enfoques de inversión, cuyo objetivo central será la recopilación y evaluación de rendimientos a lo largo del tiempo, derivados de la aplicación del trading algorítmico sobre el índice MERVAL. La implementación técnica se llevará a cabo mediante el lenguaje de programación Python, utilizando principalmente la biblioteca pyRofex, diseñada para interactuar con los mercados financieros operados por Rofex.

Parte I: Consideraciones sobre PyRofex y APIs WebSocket

Consideraciones Generales

En la presente sección se expondrán, de manera sintética, una serie de lineamientos generales que serán aplicables de forma transversal a todos los proyectos desarrollados. Estas directrices constituyen el marco operativo común sobre el cual se estructuran los distintos algoritmos, permitiendo homogeneidad y coherencia en su implementación.

En primer lugar, se abordarán aspectos técnicos vinculados al uso de APIs basadas en el protocolo websocket, enfatizando las implicancias de su funcionamiento en tiempo real y la necesidad de estructurar secciones específicas del código en función de esta lógica asincrónica. Esta comprensión resulta esencial para garantizar una comunicación eficiente y estable con los servidores de mercado.

Asimismo, se realizará una revisión detallada de la biblioteca pyRofex, con el propósito de comprender en profundidad su arquitectura y funcionalidades. Este análisis permitirá optimizar su utilización dentro de los proyectos, maximizando la eficiencia operativa y minimizando errores derivados del desconocimiento de sus métodos y parámetros.

Finalmente, se documentaron los errores encontrados a lo largo del desarrollo, con el objetivo de establecer un registro técnico que facilite su futura resolución y contribuya a la mejora continua del proceso de implementación.

1. ¿Qué es PyRofex?

PyRofex es una biblioteca de Python que permite la interacción con las API REST y WebSocket provistas por el mercado ROFEX. Su diseño está orientado a simplificar y agilizar el proceso de desarrollo, evitando que los programadores debían invertir una cantidad considerable de tiempo en investigar la estructura de las APIs y en codificar la lógica de conexión desde cero. De este modo, PyRofex permite a los desarrolladores concentrarse en los aspectos centrales de sus aplicaciones, como la lógica de trading, la gestión del riesgo o el análisis de datos, sin distraerse en tareas técnicas de bajo nivel relacionadas con la conectividad.

No obstante, es importante señalar que la biblioteca presenta ciertas limitaciones en cuanto a su desarrollo y robustez. Por ejemplo, en el manejo de posiciones abiertas, la información retornada por la API resulta en ocasiones poco fiable, ya que tiende a “acumular”operaciones de compra / venta sin ofrecer una identificación precisa que permita una interpretación directa y exacta de los datos. Esta deficiencia técnica impone restricciones al momento de automatizar análisis y decisiones operativas, dificultando la trazabilidad y el control efectivo de las posiciones mantenidas.

En consecuencia, el enfoque adoptado en este proyecto se centrará en el uso de las funcionalidades más básicas y estables de PyRofex, con el propósito de garantizar tanto el cumplimiento del proceso de homologación de la cuenta comitente como la correcta identificación y gestión de situaciones anómalas que puedan surgir durante la operativa. Muchas de estas inconsistencias no se manifiestan como errores explícitos, sino que requieren un seguimiento prolongado en el tiempo para su adecuada detección y diagnóstico.

La primer función que utilizaremos de PyRofex nos permite crear la conexión con la cuenta comitente previamente homologada en primary. La primera función que utilizaremos es crear la conexión con la cuenta para ser capaz de obtener información de mercado y envío de órdenes.

2. Principales funciones de PyRofex

Uno de los componentes más relevantes de la biblioteca PyRofex es su módulo de conexión a través de WebSocket, el cual permite establecer una comunicación en tiempo real con la API del mercado para recibir información de mercado, reportes de órdenes, y gestionar posibles errores o excepciones. Este módulo opera mediante una serie de handlers, o manejadores de eventos, que procesan los datos entrantes y permiten su tratamiento programático dentro de una estrategia algorítmica.

La inicialización de esta conexión se realiza mediante la función:

```
pyRofex.init_websocket_connection(market_data_handler=marketDataHandler,
                                  order_report_handler=orderReportHandler,
                                  error_handler=errorHandler,
                                  exception_handler=exceptionHandler)
```

Este fragmento de código ilustra la forma en que PyRofex permite vincular funciones personalizadas para el manejo de eventos específicos. Particularmente, el market_data_handler y el order_report_handler son fundamentales para el procesamiento de información operativa, y requieren un tratamiento cuidadoso para evitar errores lógicos en la implementación.

Uno de los aspectos más críticos al recibir datos de mercado es el tratamiento adecuado de los mensajes recibidos. Si se omiten ciertas precauciones en esta etapa, pueden producirse inconsistencias, especialmente si se combinan datos actuales con históricos, lo cual es frecuente en contextos de análisis técnico que requieren series temporales extensas.

Un ejemplo funcional para el manejo de la información del mercado podría estructurarse de la siguiente manera:

```
reporte_informacion_de_mercado = pd.DataFrame()

def marketDataHandler(message):
    global reporte_informacion_de_mercado
```

Aquí, se declara la variable reporte_informacion_de_mercado como global para asegurar que los datos capturados dentro de la función marketDataHandler sean accesibles desde el contexto principal del programa. En caso contrario, aunque los datos lleguen correctamente desde la API, no se reflejarán en la estructura de datos que gestiona el flujo principal de información.

El mensaje recibido por el handler debe desglosarse cuidadosamente para extraer los elementos clave, tal como se presenta a continuación:

```
if len(message) > 0:
    ticker = message["instrumentId"]["symbol"]
    close_price = message["marketData"]["LA"]["price"]
    bi_price = message["marketData"]["BI"][0]["price"]
    of_price = message["marketData"]["OF"][0]["price"]
    timemstamp = message["timestamp"]

        mensaje_informacion_de_mercado = {"Ticker": ticker,
                        "Close": close_price,
                        "bid": bi_price,
                        "offer": of_price,
                        "Datetime": timemstamp}

            informacion_de_mercado.append(mensaje_informacion_de_mercado)
            reporte_informacion_de_mercado = pd.DataFrame(informacion_de_mercado)
                                        reporte_informacion_de_mercado["Datetime"]      =
        pd.to_datetime(reporte_informacion_de_mercado["Datetime"],
                                                unit="ms",
                                                errors="coerce")

            return reporte_informacion_de_mercado
```

El uso de la condición if len(message) > 0: resulta esencial para garantizar que el mensaje recibido contenga efectivamente información válida. En ausencia de esta verificación, podrían cargarse datos nulos o vacíos, lo cual comprometería la integridad de las bases de datos construidas a partir de estos registros.

Finalmente, es crucial comprender que el diseño de la arquitectura de manejo de datos debe centrarse en conservar únicamente el dato más reciente de cada activo solicitado, evitando duplicaciones o inconsistencias que dificulten el análisis posterior. Este enfoque asegura la construcción de una base de datos sólida, precisa y coherente para el desarrollo de estrategias de trading algorítmico.

2.1 Manejo de las bases de información de mercado.

Para comprender con precisión la lógica con la que se gestionarán los datos en los futuros desarrollos, resulta indispensable realizar un esquema conceptual sobre el funcionamiento del flujo de información desde la API y su almacenamiento en bases estructuradas. Este enfoque busca anticipar problemas comunes que podrían surgir debido a supuestos erróneos o al desconocimiento del comportamiento por defecto de la API de ROFEX.

Cuando se establece la conexión mediante PyRofex, la información de mercado es recibida en forma de mensajes streaming que se actualizan constantemente, pero no reemplazan ni eliminan los datos previos. Esto implica que los valores se acumulan progresivamente, y en muchos casos pueden incluir duplicaciones o información redundante si no se aplican filtros o mecanismos de control adecuados. Este comportamiento, de no ser adecuadamente comprendido, puede generar inconsistencias graves en el tratamiento de los datos, especialmente al construir indicadores técnicos o estrategias automatizadas.

Tal como se explicó en la sección anterior, en esta instancia únicamente se solicita al market data handler información crítica: el símbolo del instrumento (symbol), el último precio negociado (LA), el mejor precio de compra (BI) y de venta (OF), junto con la marca temporal (timestamp) del evento. Esta selección no es arbitraria, sino que responde a limitaciones observadas en la calidad y estructura de los datos ofrecidos por la API.

Por ejemplo, aunque PyRofex permite acceder a información sobre máximos, mínimos y volumen operado, estas variables presentan ciertos inconvenientes prácticos:

- Máximos y mínimos: los valores provistos corresponden a máximos y mínimos del día bursátil completo, lo cual no se adapta correctamente al análisis basado en timeframes inferiores, como por ejemplo una hora. Si se intenta construir indicadores que dependen de estos extremos (como RSI, Bollinger Bands o ATR), los resultados serán incorrectos, ya que todos los registros en un mismo día compartirán el mismo máximo y mínimo.
- Volumen operado: en múltiples pruebas realizadas, el valor de volumen retorna consistentemente igual a cero, lo cual indica una limitación en la disponibilidad o exposición de ese dato vía WebSocket. Este comportamiento impide cualquier análisis cuantitativo confiable basado en volumen.

Dada esta situación, la estrategia adoptada consiste en registrar solamente la información confiable y consistente en una base de datos estructurada en formato SQL. Esta base contiene el último dato actualizado por cada ticker solicitado. La lógica de actualización se activa según las necesidades del sistema, y su propósito principal es el de construir estructuras tipo “vela” (candlestick) para su posterior análisis técnico.

El procedimiento se basa en tomar, para cada período de análisis:

- El primer precio registrado (que representa la apertura),
- El máximo de los precios recibidos,
- El mínimo de los precios en ese intervalo,
- Y el último precio registrado (el cierre del período).

Con esta información, se construyen las velas correspondientes al timeframe deseado, las cuales pueden utilizarse posteriormente para análisis visuales, cálculos de indicadores, o toma de decisiones algorítmicas. Esta arquitectura modular, basada en una base consolidada y confiable de precios actualizados, permite flexibilidad en el diseño del sistema sin depender de las limitaciones técnicas de la API en tiempo real.

2.2 Conexión con bases históricas.

Una vez construida la base de datos en tiempo real que almacena el último dato recibido desde la API para cada ticket, se procede a estructurar la información en forma de velas, como se detalló previamente. Este proceso implica capturar, para cada intervalo de tiempo definido (por ejemplo, 15 minutos, 1 hora, etc.), el primer valor (apertura), el máximo, el mínimo y el último (cierre).

Para que este enfoque funcione correctamente, es fundamental implementar un mecanismo que permita:

1. Almacenar los datos construidos en forma de vela una vez finalizado el intervalo.
2. Vaciar o reiniciar la estructura temporal que contenía los datos crudos del período anterior, de modo que la nueva vela comience a formarse sin arrastrar información duplicada o desfasada.

De esta manera, se garantiza que los datos en memoria correspondan exclusivamente al nuevo intervalo en formación, evitando errores o solapamientos entre velas.

Posteriormente, esta base generada en tiempo real se concatena con la base de datos histórica previamente descargada. Esta combinación debe realizarse bajo ciertas condiciones estrictas:

- Ambas fuentes deben coincidir en el timeframe utilizado.
- El formato de las columnas (por ejemplo, nombres, tipos de datos y formato de fechas) debe ser idéntico.
- Se debe verificar que no se dupliquen filas si alguna vela de la base en tiempo real ya está presente en la base histórica.

El resultado de esta operación es una nueva base de datos compuesta, que integra:

1. La información histórica previamente descargada (Esta información estará descargada en formato SQL).
2. La base de velas generadas en tiempo real con los datos recolectados desde la API durante la jornada.
3. La vela actualmente en formación, que se alimenta con los datos entrantes en el instante actual y se sumará como nueva fila al finalizar su intervalo correspondiente.

Esta arquitectura compuesta permite realizar análisis técnico en tiempo real sobre una base consolidada y actualizada, manteniendo coherencia histórica y operativa. Resulta especialmente útil para sistemas algorítmicos que requieren precisión temporal, como los que aplican indicadores técnicos o estrategias de backtesting sobre series temporales extendidas.

Una consideración fundamental al trabajar con los datos provenientes de la API de PyRofex es el formato de las celdas descargadas, ya que toda la información llega inicialmente en formato entero (int), incluyendo el campo correspondiente al tiempo.

Este último punto resulta especialmente relevante, ya que el valor temporal se entrega en formato timestamp (milisegundos desde el epoch, es decir, desde el 1 de enero de 1970). Si no se realiza una conversión adecuada a un formato de fecha y hora reconocible, estos valores se interpretarán erróneamente como pertenecientes al 1 de enero de 1970, fecha por defecto del sistema UNIX cuando no se establece un valor válido.

Este problema tiene consecuencias directas al momento de ordenar la información cronológicamente. Si se concatenan datos históricos con los datos en tiempo real extraídos desde la API sin tratar correctamente el campo de tiempo, los registros mal convertidos (con fecha 1970-01-01) aparecerán al comienzo de la base de datos al ordenar de forma descendente por fecha, generando un desfase temporal que puede afectar los análisis, la construcción de velas, y cualquier tipo de cálculo dependiente de la secuencia temporal.

3. Manejo de órdenes

En el contexto del manejo de órdenes dentro de la plataforma PyRofex, se utiliza la función denominada order_report_handler. Esta función se activa cada vez que se genera un evento relacionado con una orden, permitiendo así el seguimiento en tiempo real de las operaciones que el bot ejecuta a lo largo de la jornada. Esta característica resulta particularmente útil para realizar actualizaciones dinámicas del portafolio, ya que los cambios generados por la ejecución de órdenes pueden reflejarse de forma inmediata en la estructura del mismo.

Dado que uno de los objetivos es mantener un registro organizado y actualizado de los instrumentos operados, se identificó que la estrategia más eficiente consiste en aprovechar directamente la información brindada por la función order_report_handler, en lugar de recurrir a otros métodos que PyRofex ofrece para consultar el estado del portafolio.

El empleo exclusivo de esta función se justifica en base a diversos períodos de prueba con distintos bots, en los cuales se observó que los mensajes capturados por orderReportHandler se actualizan de forma más eficiente y rápida que las consultas realizadas mediante otros métodos de la API. Además, esta implementación ofrece una mayor claridad respecto del estado actual del portafolio, ya que permite identificar en tiempo real qué activos se poseen, en qué cantidad, y bajo qué condiciones fueron adquiridos. También posibilita estructurar y analizar esta información mediante herramientas como SQL, facilitando el registro y procesamiento de grandes volúmenes de datos históricos.

A continuación se presenta un ejemplo del código utilizado para extraer la información relevante a partir de los mensajes capturados por order_report_handler en el método previamente visto de pyRofex.init_websocket_connection a través de una función llamada orderReportHandler:

```
def orderReportHandler(message):
    ticker = message['orderReport']["instrumentId"]["symbol"]
    tipo_orden = message['orderReport']["ordType"]
    timeStamp = message['orderReport']['transactTime']
    side = message['orderReport']["side"]
    status = message['orderReport']['status']
    cantidad = message['orderReport']["orderQty"]
    precio = message['orderReport']["price"]
    orderId = message['orderReport']['orderId']
    clienteId = message["orderReport"]["clOrdId"]
    propietario = message["orderReport"]["proprietary"]
```

Este fragmento permite capturar los datos esenciales de cada orden: el instrumento operado, el tipo y lado de la orden, el momento de ejecución, el estado actual, el volumen y precio de la orden, así como su identificación única tanto interna como externa.

4. Otras funciones de PyRofex

…

Backtest

El primer paso fundamental en el desarrollo de cualquier sistema automatizado de trading es la realización de pruebas retrospectivas o backtesting. Esta técnica consiste en evaluar el comportamiento histórico de una estrategia de inversión, utilizando datos pasados del mercado para estimar su rendimiento potencial, su robustez frente a diferentes condiciones de mercado, y su viabilidad operativa en un entorno real.

Con el propósito de facilitar este proceso, se desarrollará un módulo capaz de ejecutar backtests sobre cualquier conjunto de datos que se le proporcione, permitiendo la configuración de los parámetros mediante entradas dinámicas (inputs). Este enfoque garantiza que las pruebas puedan adaptarse fácilmente a diferentes marcos temporales, activos financieros y configuraciones estratégicas.

Cabe destacar que el diseño de este módulo se apoya en desarrollos previos elaborados en otros proyectos, los cuales serán adaptados a las nuevas necesidades planteadas en el presente documento. Asimismo, dado que el objetivo general es implementar bots de trading que permitan observar y analizar su comportamiento a lo largo del tiempo, los componentes desarrollados estarán estructurados de manera flexible y escalable, facilitando así su reutilización y extensión en futuros experimentos o aplicaciones reales.

Bot I: CAPM Dinámico.

El primer algoritmo desarrollado se fundamenta en el modelo de valoración de activos financieros conocido como CAPM (Capital Asset Pricing Model), haciendo especial énfasis en el coeficiente beta (β), el cual representa la sensibilidad de un activo individual frente a los movimientos del mercado en general, en este caso, el índice MERVAL. La propuesta de este bot parte de un concepto ampliamente difundido en el ámbito financiero, pero introduce una mejora sustancial mediante su automatización y dinamismo, lo cual permite una operativa más eficiente y adaptativa.

El mecanismo de funcionamiento consiste en ejecutar órdenes de compra o venta de activos en función de su variación porcentual desde el momento en que el bot inicia las operaciones. Para ello, se toma como referencia el precio inicial de cada activo y se mide su evolución relativa, ajustando dicha variación de acuerdo con su beta estimada. Esto permite identificar qué tan desfasado se encuentra un activo con respecto al comportamiento esperado según su exposición sistemática al riesgo de mercado.

Cabe señalar que este enfoque se inspira en una estrategia tradicional de inversión de largo plazo, la cual establece umbrales fijos —por ejemplo, adquirir activos tras una caída del 5% y venderlos ante un incremento del 10%—. Sin embargo, dichos parámetros carecen de adaptabilidad, lo que reduce la eficiencia del modelo en entornos volátiles. Mediante el uso del trading algorítmico, se pueden establecer umbrales dinámicos ajustados a las características individuales de cada activo, permitiendo realizar compras acumulativas, liquidaciones parciales o cierres completos de posición de manera estratégica y automatizada.

Dado que se trata de una estrategia de acumulación, se considera el precio promedio ponderado de compra para evaluar si la posición abierta está generando rendimientos positivos, lo cual resulta esencial para tomar decisiones de gestión de capital coherentes con los objetivos de rentabilidad y riesgo del inversor.

Con el objetivo de simplificar las operaciones y facilitar el análisis de resultados, todas las órdenes de compra se realizarán por una única unidad de acción. Esta decisión metodológica responde a la necesidad de representar de forma precisa los rendimientos generados por la estrategia sin requerir un volumen de capital significativo. Al mantener constante la cantidad adquirida en cada transacción, se posibilita una acumulación progresiva de activos, permitiendo una evaluación clara y desagregada del desempeño de la estrategia a lo largo del tiempo.

Bibliografía

Bacidore, J. M. (2020). Algorithmic trading: A practitioner 's guide. TBG Press.

Chang, E. P. (s.f.). Algorithmic trading & winning strategies.

Halls-Moore, M. L. (s.f.). A step-by-step guide to quantitative strategies: Successful algorithmic trading – Applying the scientific method for profitable trading results.

Hilpisch, Y. (2020). Python for algorithmic trading: From idea to cloud deployment. O'Reilly Media.

McKinney, W. (2018). Python for data analysis (2nd ed.). O'Reilly Media.

Tuckfield, B. (2021). Dive into algorithms. No Starch Press.

Beaulieu, A. (2020). Learning SQL (3rd ed.). O'Reilly Media.

Ivanov, I. (2015). NumPy beginner 's guide (3rd ed.). Packt Publishing.

Primary API. (s.f.). Primary API reference manual. Disponible en: https://apihub.primary.com.ar/assets/docs/Primary-API.pdf