Extensión

StrategyQuant X

Guía de Programación (en curso)

Traducido por: **Darío Redes**

Version X Build 109 *Last updated: 30.8.2018*

Soporte

Si tiene un problema con algunas cuestiones y necesita ayuda, o simplemente tiene cualquier pregunta (relacionada con el sistema), recuerde que su compra también incluye un soporte.

Estamos aquí para usted, se puede poner en contacto con nosotros usando nuestro foro:

www.StrategyQuant.com/forum

La sección de artículos en nuestro sitio web provee una fuente adicional de conocimiento

docs.StrategyQuant.com

Copyright

Advertencia de riesgo

Declaración de Advertencia de riesgo

Este Manual no es ninguna solicitud, ni una oferta de Comprar/Vender cualquier producto financiero. Los contenidos de este Manual se utilizan exclusivamente con objetivos informativos generales.

Aunque cada prueba ha sido hecha para asegurar la exactitud, el autor no da ninguna garantía implícita o expresa en cuanto a su exactitud. El autor no acepta ninguna responsabilidad de error u omisión. Todos los ejemplos son proporcionados con objetivos ilustrativos solamente y no deberían ser interpretados como un asesoramiento sobre inversiones.

No se está realizando ninguna representación de que cualquier cuenta u operador puedan conseguir ganancias o pérdidas similares a las discutidas en este Manual. El rendimiento pasado no puede ser considerado como un indicativo de rendimientos futuros.

La información proporcionada en este Manual no está destinada a su distribución o uso por parte de ninguna persona o entidad en ninguna jurisdicción o país donde tal distribución o uso sea contrario a la ley o regulación que someta al autor a cualquier requisito de registro dentro de dicha jurisdicción o país.

Los resultados de rendimiento hipotéticos tienen muchas limitaciones inherentes, algunas de las cuales se mencionan a continuación. No se está realizando ninguna representación de que cualquier cuenta logre o probablemente pueda conseguir ganancias o pérdidas similares a los mostrados. De hecho, con frecuencia existen agudas diferencias entre los resultados de rendimiento hipotéticos y los resultados reales logrados posteriormente por cualquier sistema de trading particular.

Una de las limitaciones de los resultados del rendimiento hipotético es que generalmente se preparan con el beneficio del backtesting. Además, el trading hipotético no implica un riesgo financiero y ningún registro hipotético puede explicar completamente el impacto del riesgo financiero en el trading real.

Por ejemplo, la capacidad de soportar las pérdidas o de adherirse a un programa de trading particular a pesar de las pérdidas comerciales es importante, lo que también pueden afectar negativamente los resultados comerciales. Existen numerosos factores relacionados con el mercado en general o con la implementación de cualquier programa de negociación específico, que no puede ser pueden contabilizar por completo en la preparación de los resultados de rendimiento hipotéticos. Todo lo cual puede afectar negativamente a los resultados comerciales reales.

Descargo de responsabilidad exigido por el gobierno de EE. UU. - Commodity Futures Trading Commission. El Comercio de Futuros, Divisas y Opciones tiene grandes recompensas potenciales, pero también un gran riesgo potencial. Debe ser consciente de los riesgos y estar dispuesto a aceptarlos para invertir en los mercados de futuros y opciones. No negocie con dinero que no pueda permitirse perder. Esto no es una solicitud ni una oferta para comprar / vender futuros u opciones. No se está haciendo ninguna representación de que cualquier cuenta logre o pueda obtener ganancias o pérdidas similares a las discutidas en este sitio web. El desempeño anterior de cualquier sistema de comercio o metodología no es necesariamente indicativo de resultados futuros.

ARTÍCULO 4.41 DE CFTC - LOS RESULTADOS HIPOTÉTICOS O SIMULADOS DE RENDIMIENTO TIENEN CIERTAS LIMITACIONES. A MENOS QUE SEA UN REGISTRO DE RENDIMIENTO REAL, LOS RESULTADOS SIMULADOS NO REPRESENTAN EL COMERCIO REAL. POR LO TANTO, YA QUE LAS OPERACIONES NO HAN SIDO EJECUTADAS, LOS RESULRTADOS PUEDEN HABER COMPENSADO POR DEBAJO O ENCIMA POR EL IMPACTO GENERADO, EN CASO DE CIERTOS FACTORES DE MERCADO, COMO LA FALTA DE LIQUIDEZ. LOS PROGRAMAS DE COMERCIALIZACIÓN SIMULADOS EN GENERAL TAMBIÉN ESTÁN SUJETOS AL HECHO DE QUE ESTÁN DISEÑADOS CON EL BENEFICIO DE LA DEFICINICIÓN. NO SE REALIZA NINGUNA REPRESENTACIÓN DE QUE CUALQUIER CUENTA HAYA O PUEDA LOGRAR BENEFICIOS O PÉRDIDAS SIMILARES A LAS EXPRESADAS.

Renuncia a la Garantía

Usted acepta utilizar este programa bajo su propio riesgo.

El software StrategyQuant podría fallar o no funcionar correctamente. Todo software está sujeto a errores de programación involuntarios y errores de programación introducidos en el código que comprende ese software. Cualquiera de estos errores y errores de programación puede causar que en el software en el cual están localizados para fallar o no trabajar correctamente. La aplicación de StrategyQuant y las estrategias comerciales generadas por el programa están sujetas a este riesgo. A pesar de las pruebas, los errores involuntarios y los errores de programación todavía pueden causar un fracaso en la estrategia comercial, causando errores en el trading.

TIENE QUE SER CONSCIENTE DE QUE LAS ESTRATEGIAS DE NEGOCIACIÓN AUTOMÁTICAS PODRÍAN FALLAR POR CUALQUIER MOTIVO Y PODRÍAN RESULTAR EN LA PÉRDIDA DE TODO EL DINERO QUE HA DEPOSITADO EN SU CUENTA DE BROKERAGE QUE UTILIZA PARA EL TRADING EN DIRECTO BASADO EN LOS ALGORITMOS GENERADOS POR StrategyQuant. DEBERÍA DISCUTIR CON UN PROFESIONAL DE INVERSIONES LOS RIESGOS DEL TRADING EN GENERAL Y DEL TRADING ALGORÍTMICO EN PARTICULAR. SI UTILIZA CUALQUIER ALGORITMO EN EL TRADING EN DIRECTO BAJO SU PROPIO RIESGO, ES SU OBLIGACIÓN ANALIZAR Y PROBAR CUALQUIER ALGORITMO COMERCIAL ANTES DE PONERLO EN PRODUCCIÓN Y MONITOREAR CONTINUAMENTE LA OPERACIÓN DEL ALGORITMO COMERCIAL EN LA PRODUCCIÓN PARA GARANTIZAR QUE FUNCIONA CORRECTAMENTE Y EN CUMPLIMIENTO CON CUALQUIER REGLAMENTO APLICABLE.

RENUNCIA DE GARANTÍA. EL SOFTWARE SE PROPORCIONA "TAL CUAL", SIN GARANTÍA DE NINGÚN TIPO, INCLUYENDO, SIN LIMITACIÓN, LAS GARANTÍAS DE COMERCIABILIDAD E IDONEIDAD PARA UN PROPÓSITO EN PARTICULAR. USTED ASUME EL RIESGO TOTAL EN CUANTO A LA CALIDAD Y EL RENDIMIENTO DEL SOFTWARE QUE ADQUIERE.

LIMITACIÓN DE RESPONSABILIDAD. BAJO NINGUNA CIRCUNSTANCIA Y BAJO NINGUNA TEORÍA LEGAL, AGRAVIO, CONTRATO, O DE OTRO MODO, EL AUTOR O SUS PROVEEDORES O REVENDEDORES SERÁN RESPONSABLES ANTE USTED O CUALQUIER OTRA PERSONA POR CUALQUIER DAÑO INDIRECTO, ESPECIAL, INCIDENTAL O CONSECUENCIAL O PUNITIVO DE CUALQUIER CARÁCTER INCLUYENDO, SIN LIMITACIÓN, DAÑOS POR PÉRDIDA DE FIDEICOMISO, PÉRDIDA DE TRABAJO, FALLA DE LA COMPUTADORA O MAL FUNCIONAMIENTO, O CUALQUIER OTRA PÉRDIDA O DAÑO COMERCIAL. EN NINGÚN CASO EL AUTOR SERÁ RESPONSABLE POR NINGÚN DAÑO QUE EXCEDA EL PRECIO DE LA LISTA DEL AUTOR PARA UNA LICENCIA AL SOFTWARE.

INCLUSO EL AUTOR HA SIDO INFORMADO DE LA POSIBILIDAD DE DICHOS DAÑOS, O DE CUALQUIER RECLAMO POR CUALQUIER OTRA PARTE. ESTA LIMITACIÓN DE RESPONSABILIDAD NO SE APLICARÁ A LA RESPONSABILIDAD POR MUERTE O LESIONES PERSONALES EN LA MEDIDA EN QUE LA LEY APLICABLE PROHIBE DICHA LIMITACIÓN. ADEMÁS, ALGUNOS ESTADOS NO PERMITEN LA EXCLUSIÓN O LIMITACIÓN DE DAÑOS INCIDENTALES O CONSECUENCIALES, POR LO QUE ESTA LIMITACIÓN Y EXCLUSIÓN PODRÍA NO APLICARSE EN SU CASO.

Tabla de Contenidos

1 Introducción 6
1.1 Extensión StrategyQuant X 6
2 Indicadores y Señales – Bloques de Construcción 8
2.1 Agregando nuevos indicadores a StrategyQuant X 8
2.1.1 Paso 1 – Creando un nuevo indicador personalizado en Code Editor 8
2.1.2 Paso 2 – Modificar la plantilla por defecto e implementar el indicador 13
2.2 Creando una nueva señal basada en el nuevo indicador 18
2.2.1 Paso 1 – Crear una nueva señal en Code Editor 18
3. Valores estadísticos y columnas 23
3.1 Agregar nueva columna al banco de datos (valor estadístico) 23
3.1.1 NetProfit-ejemplo de valor estadístico del cálculo de los trades 23
3.1.2 Ret/DD Ratio – ejemplo de cálculo de valores estadísticos de otros valores 24
3.2 Nueva columna Databank: relación entre valor principal y verificación cruzada avanzada .25
3.2.1 Paso 1 - Iniciar CodeEditor y crear un nuevo fragmento 25
3.2.2 Paso 2 - Agregar cálculo de valor al fragmento 27
3.2.3 Paso 3 - Compile los fragmentos y reinicie SQ 28
3.2.4 Paso 4 – Usar la nueva columna del Banco de datos 29

1 Introducción

1.1 Extensión StrategyQuant X

StrategyQuant versión X fue creada desde cero como una plataforma abierta y ampliable.

La mayor parte de la funcionalidad se implementa utilizando complementos o fragmentos de código. Cual es la diferencia entre ellos:

- **Plugin** – es un módulo más grande que incluye una interfaz de usuario y un código de fondo. Un ejemplo de complemento es toda la pantalla del Builder, y contiene otros sub-complementos: cada pestaña de configuración y cada pestaña de resultados son otro complemento.

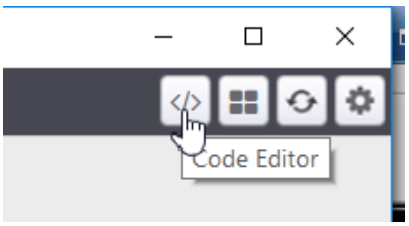
  El desarrollo de complementos no está cubierto en este breve manual, es muy técnico y complejo y posiblemente se ofrecerá en algunas versiones futuras de StrategyQuant X.

- **Snippet** – Es una "función" implementando una cosa. Por ejemplo, cada modelo de gestión del dinero es un fragmento. Cada indicador y bloque de construcción es un fragmento. Cada columna del banco de datos es un fragmento.

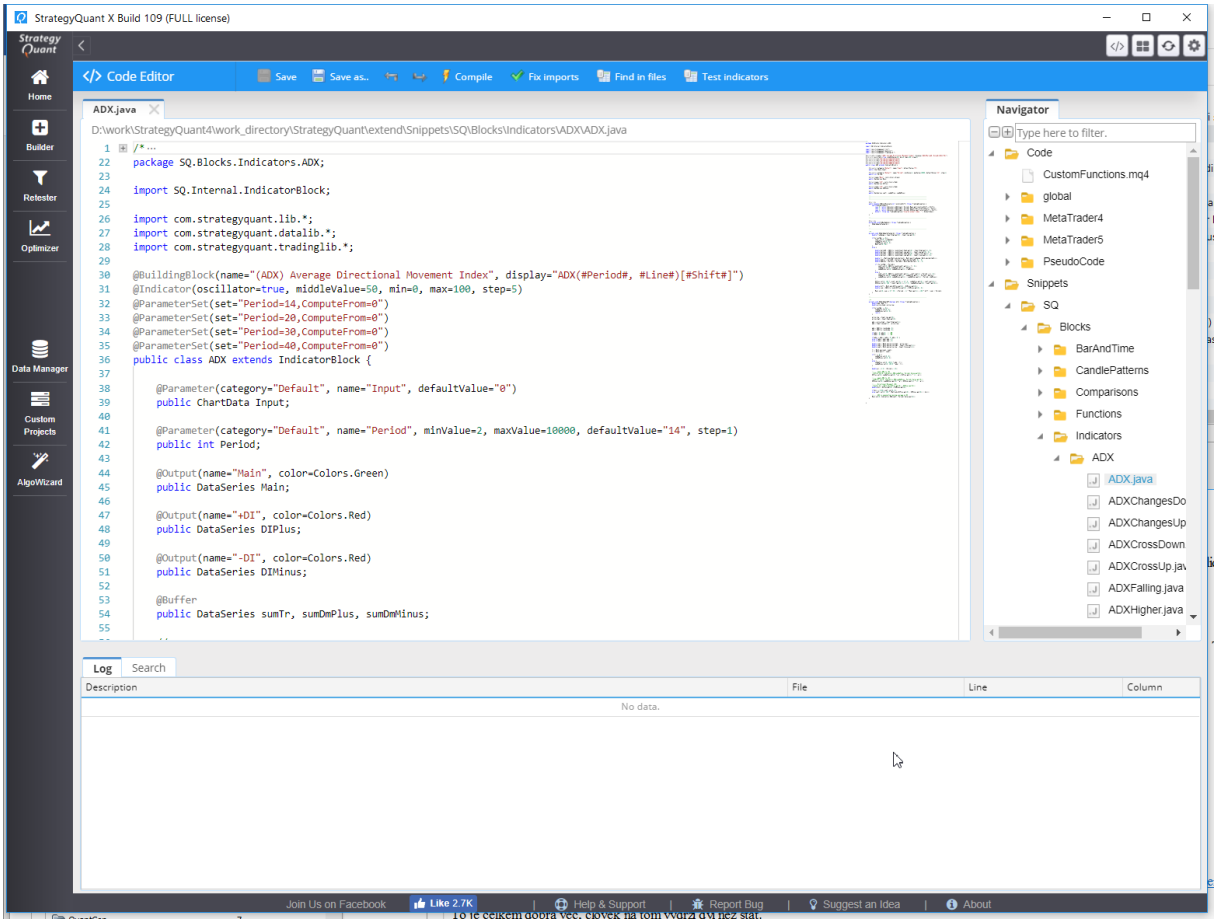
  Esto te permite ampliar StrategyQuant con tus propios indicadores, valores estadísticos, etc. de una manera relativamente simple.

Cada fragmento de código es una clase Java corta que implementa alguna función. Daremos un ejemplo de los fragmentos más comunes en este manual para que puedas comenzar a usarlos.

Se puede acceder a los fragmentos a través del ícono de CodeEditor en la esquina superior derecha.



Esto abrirá el panel CodeEditor, donde puedes ver, editar y crear fragmentos (snippets).



En el lado derecho del editor puedes ver la estructura de árbol de todos los fragmentos de código SQ.

2 Indicadores y Señales – Bloques de Construcción

2.1 Agregando nuevos indicadores a StrategyQuant X

En este artículo, iremos agregando un nuevo indicador personalizado a StrategyQuant X. A diferencia de SQ3, SQ X permite extender el programa al crear tus propios indicadores y bloques de construcción personalizados, de manera similar a cómo puedes agregar nuevos indicadores a las plataformas de operaciones normales como MetaTrader 4/5, Tradestation, etc.

Pasaremos por este proceso paso a paso para explicar todo lo que es necesario.

Al final, tendremos un nuevo indicador personalizado agregado a StrategyQuant y podrás agregarlo a la lista de componentes básicos para la generación de estrategias.

Usaremos el indicador Force Index como ejemplo. Este indicador ya está incorporado en Metatrader, puedes encontrar su código fuente aquí: https://www.mql5.com/en/code/8013

Así es como se ve el indicador en el gráfico:

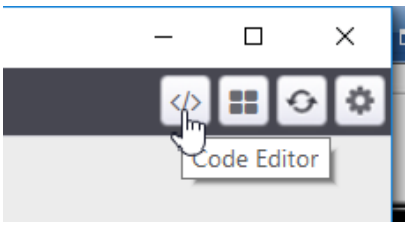
2.1.1 Paso 1 – Creando un nuevo indicador personalizado en Code Editor

El primer paso es crear nuestro indicador en el Editor de código. StrategyQuant utiliza internamente el lenguaje de programación Java, y los indicadores personalizados deben programarse en Java.

No puede tomar su código MQL y simplemente copiarlo y pegarlo en StrategyQuant, no funcionaría.

Debes volver a escribir el indicador en el lenguaje Java.

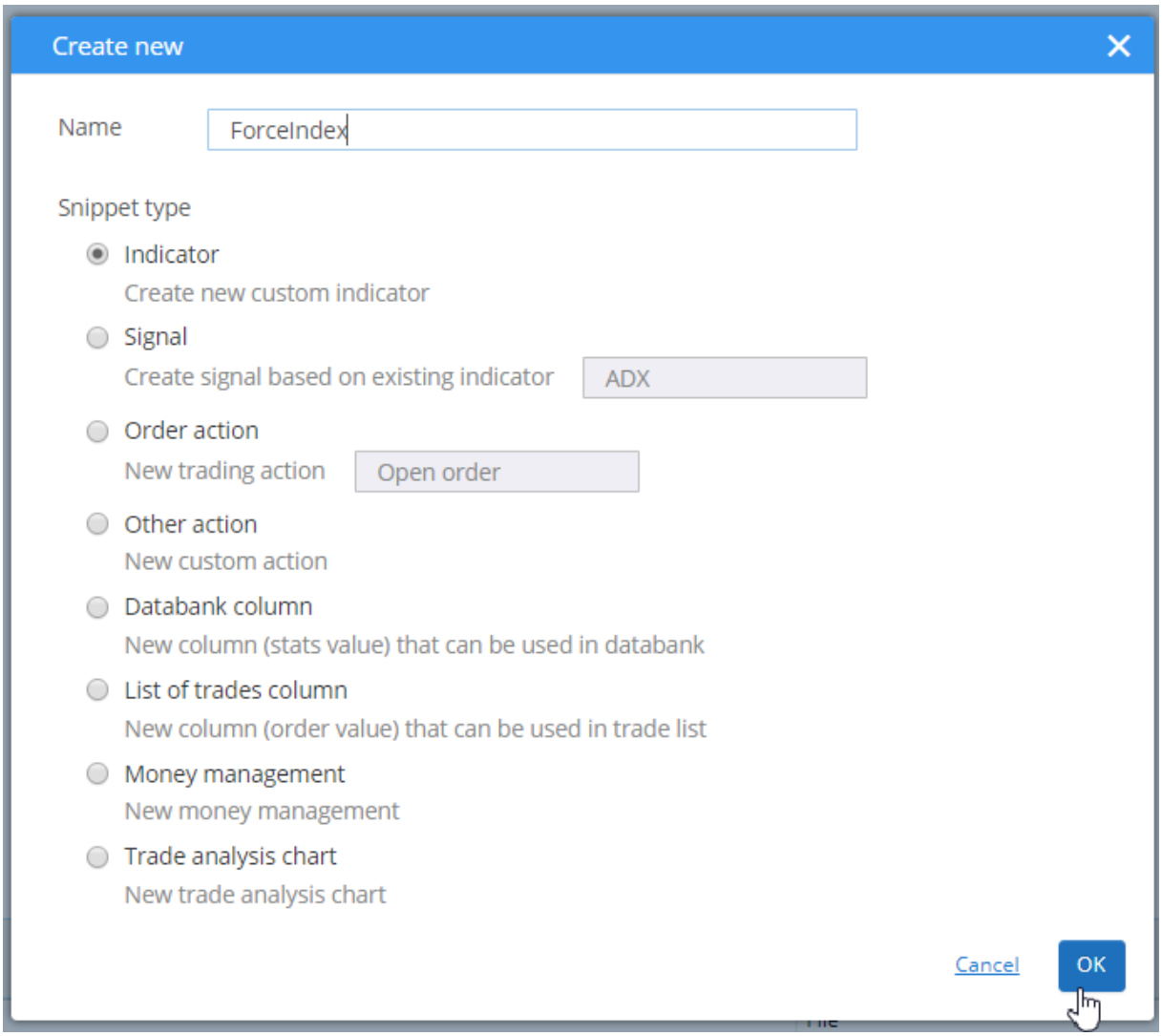
El primer paso es cambiar al Editor de Código:



Esto abrirá el Editor de código que te permite crear, editar y modificar fragmentos de código. Nuestro nuevo indicador es un tipo de fragmento.

Una vez en el Editor de código, hacemos clic en el botón Crear nuevo en la barra de herramientas superior.

Esto abrirá un diálogo, donde podemos nombrar nuestro nuevo fragmento de código y elegir su tipo. Lo llamaremos "ForceIndex" y elegiremos el indicador como un tipo de fragmento de código.



Después de hacer clic en Aceptar, se creará una nueva carpeta **ForceIndex** en Snippets -> SQ -> Bloques -> Indicadores y un nuevo archivo de fragmento de código **ForceIndex .java** en esta carpeta.

La convención en StrategyQuant es que cada indicador está en su propia carpeta, ya que es posible que más adelante deseemos crear señales para este indicador y que todos los fragmentos relacionados estén en la misma carpeta.

Esto crea un nuevo archivo para nuestro indicador ForceIndex y lo abre en el editor. Puedes ver que el indicador es una clase y ya tiene alguna estructura.

Cada indicador se extiende desde la clase IndicatorBlock y debe implementar dos métodos:

- **OnBarUpdate()** – método donde el valor del indicador se calcula y almacena en uno de los buffers de salida. Es llamado para cada barra en el gráfico.

- **OnBlockEvaluate()** – Método que es llamado cuando este indicador es evaluado por el motor de trading. Debe devolver el valor del indicador, típicamente llamando Indicators.INDICATOR_NAME(parameters).OUTPUT_BUFFER_NAME.get()

Cuando verifiques el código fuente, deberías notar algunas cosas.

Primero, los fragmentos en StrategyQuant usan mucho las anotaciones (@Parameter, @Output, @BuildingBlock) - estas anotaciones se usan para declarar una propiedad especial de una variable o clase dada - que la variable es un parámetro público o un valor de salida, o que la clase es Un bloque indicador.

En segundo lugar, una vez que crees y compiles un indicador, puedes llamarlo desde otros métodos o indicadores usando **Indicators.YourIndicator(YourIndicatorParameters).** Así es como se llama a nuestro nuevo indicador en el método **OnBlockEvaluate()**

Revisaremos el código fuente de la clase de indicador predeterminada creada a partir de la plantilla paso a paso:

```
package SQ.Blocks.Indicators.ForceIndex;

import com.strategyquant.lib.*;
import com.strategyquant.datalib.*;
import com.strategyquant.tradinglib.*;

import SQ.Internal.IndicatorBlock;
```

Esta es la declaración estándar de Java de un paquete e importaciones de clases requeridas, las que usamos en nuestra propia clase.

```
@BuildingBlock(name="(XXX) ForceIndex ", display="ForceIndex (#Period#)[#Shift#]",
returnType = ReturnTypes.Price)
@Help("ForceIndex  help text")
```

Definición anotada de nuestra clase de indicador, que indica al sistema que es un bloque de construcción con un nombre de pila.

El campo **name** es lo que se muestra en la interfaz de usuario al elegir bloques de construcción.

El campo de **display** es lo que se muestra en Wizard con parámetros. Puedes controlar qué parámetros se muestran y en qué lugar

returnType es un tipo de indicador, dice qué tipo de valor calcula este indicador. Se utiliza para que StrategyQuant sepa qué tipos deben compararse con qué. Por ejemplo, no compararía CCI (que devuelve un número) con las bandas de Bollinger (que devuelve el precio).

Hay básicamente tres tipos de retorno que un indicador puede tener:

- **Price** - el indicador calcula el precio y se muestra en el gráfico de precios, como las Bandas de Bollinger, la Media Móvil, etc.
- **Number** - el indicador calcula el número que se muestra en el propio gráfico, como CCI, RSI, MACD, etc.
- **PriceRange** - el indicador calcula el rango de precios (diferencia entre dos precios), como ATR.

Otros tipos de retorno se utilizan en otros tipos de bloques de construcción.

En nuestro caso, ForceIndex NO se muestra en el gráfico de precios, por lo que su tipo de devolución es Número, no Precio. Lo cambiaremos en el siguiente paso.

```
@Parameter
public DataSeries Input;

@Parameter(defaultValue = "10", isPeriod = true, minValue=1, maxValue=10000, step=1)
public int Period;

@Output
public DataSeries Value;
```

Lo que sigue son parámetros del indicador. Cada indicador puede tener múltiples parámetros de entrada, la plantilla crea solo dos de ellos como ejemplo. Cada parámetro se anota con la anotación @Parameter, que podría tener varios atributos.

El primer parámetro es **Input**, es la matriz de la serie de datos a partir de la cual se calcula el indicador; puede ser, por ejemplo, el precio de Apertura, Máximo, Mínimo o Cierre.

El segundo parámetro es **Period**, los indicadores suelen tener algún período en el que se calculan.

La tercera variable es **Value**, ten en cuenta que tiene una anotación diferente @Output. Esto significa que esta variable no es un parámetro indicador sino su búfer de salida. Los indicadores generalmente

tienen un solo búfer de salida, pero pueden tener más, por ejemplo, la banda de Bollinger tiene un búfer superior e inferior.

Hay un parámetro oculto más **Shift** - está predeterminado en cada indicador y le dice al motor de trading qué valor de devolución debe buscar. Por lo general, no necesitas preocuparte por este parámetro, se usa automáticamente.

Entonces hay dos métodos:

```
protected void OnBarUpdate()
```

Este es el método donde se calculan los valores del indicador. SQ lo llama internamente para cada barra y debe calcular el valor del indicador para esta barra y guardarlo en el búfer de salida.

```
public double OnBlockEvaluate(int relativeShift)
```

Este es un método al que llama el motor de backtesting cuando intenta obtener el valor del indicador. Por defecto, debería devolver el valor del indicador usando la llamada estándar:

```
return Indicators.ForceIndex(Input,Period).Value.get(relativeShift + Shift);
```

Cada indicador compilado estará disponible como un método en la clase de Indicadores, y se puede llamar usando
Indicators.INDICATOR_NAME(parameters).OUTPUT_BUFFER_NAME.get()

Este es el código fuente de la plantilla de indicador estándar. En el siguiente paso, mostraremos los cambios que deben realizarse para implementar nuestro indicador ForceIndex.

2.1.2 Paso 2 – Modificar la plantilla por defecto e implementar el indicador

El indicador creado en el paso 1 es una plantilla de indicador estándar, aún no calcula ForceIndex. Para implementar ForceIndex debemos hacer algunas cosas:

Actualizar su anotación @BuildingBlocks

Actualizaremos la anotación de este indicador de la siguiente manera:

```
@BuildingBlock(name="(FI) ForceIndex ", display="ForceIndex (#Nbr_Periods#,
#Multiplier)[#Shift#]", returnType = ReturnTypes.Number)
```

Esta es la parte más simple. Solo actualizaremos el **name** del indicador y agregaremos los nuevos parámetros reales (ver más abajo) al atributo de **display**.

También cambiamos returnType a Number, porque este indicador calcula los números que se muestran en una tabla separada, no genera un valor de precio.

Definir los parámetros de ForceIndex

Lo primero que debes hacer es un poco complicado: debemos cambiar el tipo de parámetro de **Input** predeterminado. En la plantilla estándar se define como sigue:

```
@Parameter
public DataSeries Input;
```

Es un parámetro llamado **Input**, con el tipo **DataSeries**. Esto es válido para una gran parte de los indicadores que se calculan a partir de un solo precio. Por ejemplo, los indicadores CCI, RSI, etc. se suelen calcular a partir del precio de cierre.

Puedes configurarlos para que se calculen a partir de un precio diferente, por ejemplo, a partir de un precio de Apertura, pero aun así es solo una matriz de precios.

El tipo DataSeries es un tipo de matriz de valores que contiene valores para precios de Cierre, para precios de Apertura, o para precios típicos, etc.

Sin embargo, si observas el código fuente de ForceIndex MQL, verás que calcula sus valores a partir de uno de los valores de precios y de Volumen.

Para poder acceder a múltiples matrices de precios a la vez, usaremos un tipo diferente para output:

```
@Parameter
public ChartData Chart;
```

ChartData es un tipo de objeto que representa todo el gráfico. Tendrás acceso a los precios Open, High, Low, Close, Volume en el gráfico dado.

Nota rápida – elige el tipo correcto para la variable de datos Input

Si el indicador se calcula a partir de un precio, usa DataSeries.

Si se calcula a partir de varios precios – por ejemplo, High, Low, Close, etc. – usa ChartData.

Luego hay un parámetro Periodo, también podemos dejarlo sin cambios:

```
    @Parameter(defaultValue="10", isPeriod=true, minValue=2, maxValue=1000, step=1)
    public int Period;
```

El tercer parámetro es el método de moving average:

```
    @Parameter(name="Method", defaultValue="0")
    @Editor(type=Editors.Selection, values="Simple=0,Exponential=1,Smoothed=2,Linear
weighted=3")
    public int MAMethod;
```

Esto es un poco más complejo, porque definimos una lista de selección (control de cuadro combinado) como control de edición de este parámetro. Entonces, al editar en el Wizard, el usuario podrá elegir entre los valores predefinidos.

El último parámetro es el precio aplicado - precio que se debe usar en el cálculo de la media móvil:

```
    @Parameter(defaultValue="0")
    @Editor(type=Editors.Selection,
values="Close=0,Open=1,High=2,Low=3,Median=4,Typical=5,Weighted=6")
    public int AppliedPrice;
```

Ten en cuenta que utilizamos algunos atributos en la anotación @Parameter:
defaultValue establece el valor por defecto de este parámetro.

isPeriod=true le dice a SQ que este parámetro es un período; se manejan de una manera especial, en la configuración de Builder puedes configurar el valor mínimo y máximo para períodos, por lo que SQ necesita saber qué parámetros son períodos.

minValue, maxValue, step son valores mínimos y máximos que se generarán para este parámetro durante el proceso de generación aleatoria.

Definir las salidas de ForceIndex

El indicador ForceIndex solo tiene una salida, por lo que también podemos dejarlo sin cambios:

```
    @Output
    public DataSeries Value;
```

La anotación @Output significa que este es un búfer de salida para este indicador. Ten en cuenta que es del tipo **DataSeries**, lo que significa que es una matriz de valores dobles.

Implementar el método OnBarUpdate()

Si observas el código MQL de ForceIndex, verás que es bastante simple, su código MQL es:

```
int start()
  {
   int nLimit;
   int nCountedBars=IndicatorCounted();
//---- insufficient data
   if(Bars<=ExtForcePeriod) return(0);
//---- last counted bar will be recounted
   if(nCountedBars>ExtForcePeriod) nCountedBars--;
   nLimit=Bars-nCountedBars;
//---- Force Index counted
   for(int i=0; i<nLimit; i++)
      ExtForceBuffer[i]=Volume[i]*

(iMA(NULL,0,ExtForcePeriod,0,ExtForceMAMethod,ExtForceAppliedPrice,i)-

iMA(NULL,0,ExtForcePeriod,0,ExtForceMAMethod,ExtForceAppliedPrice,i+1));
//---- done
   return(0);
  }
```

Al analizar el algoritmo, podemos ver que el valor de Force Index en una vela dada se calcula como:

ForceIndex[i] = Volume[i] *(MovAvg(Period, MAMethod, AppliedPrice)[i] - MovAvg(Period, MAMethod, AppliedPrice)[i+1];

Podemos implementarlo en Java así:

```
    protected void OnBarUpdate() throws TradingException {
        double indiValue;

        indiValue = Chart.Volume.get(0) * (
           Indicators.MA(Chart.getSeries(AppliedPrice), Period, MAMethod).Value.get(0)
           - Indicators.MA(Chart.getSeries(AppliedPrice), Period, MAMethod).Value.get(1));

        Value.set(0, indiValue);
    }
```

En este caso, el indicador es bastante simple y su cómputo requiere virtualmente solamente una línea en StrategyQuant.

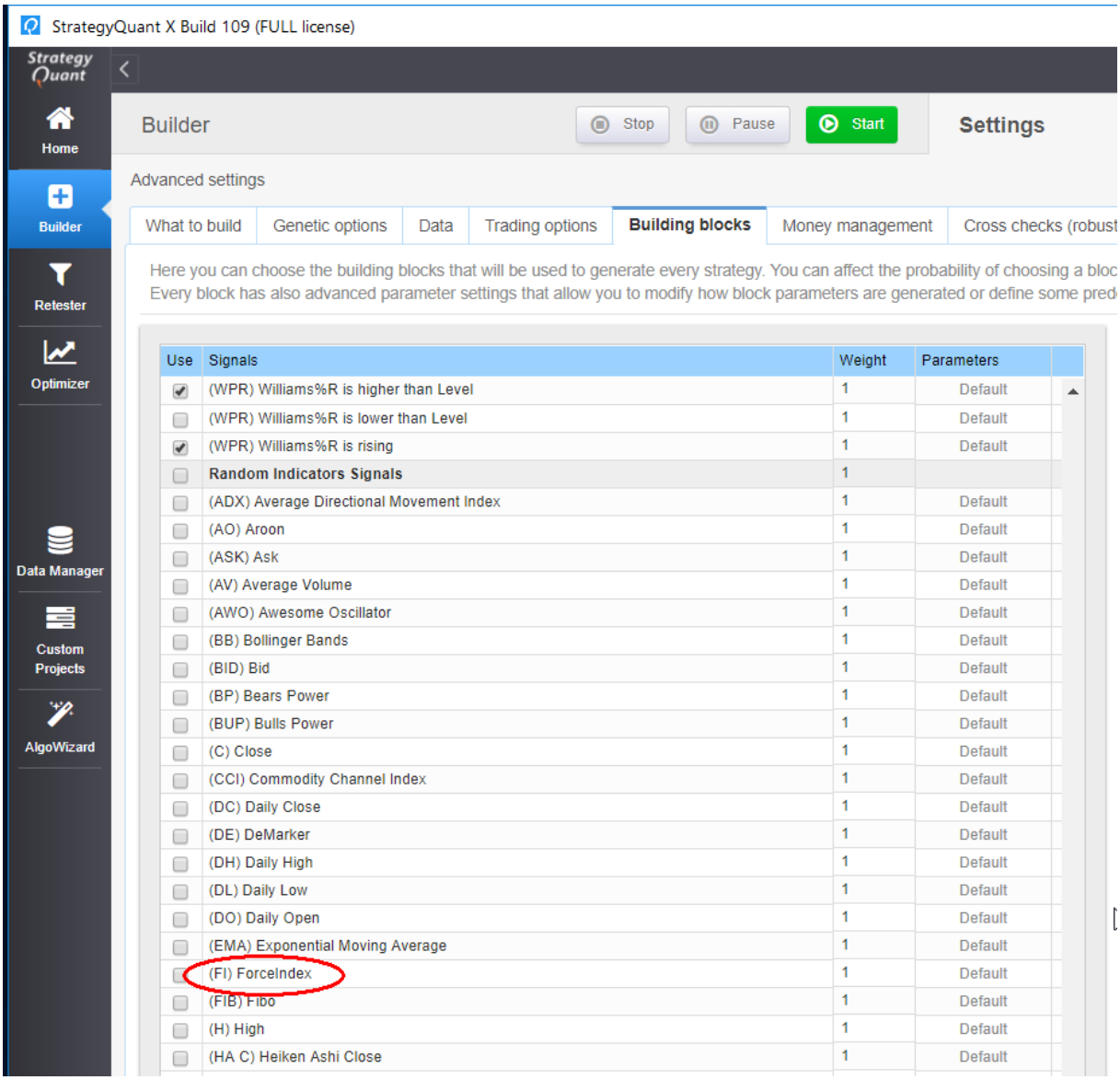
Implementar el método OnBlockEvaluate()

El segundo método que tenemos que implementar es OnBlockEvaluate (). Este método es llamado cuando StrategyQuant evalúa la condición de trading que contiene nuestro nuevo indicador ForceIndex. Es muy sencillo:

```
    public double OnBlockEvaluate(int relativeShift) throws TradingException {
        return Indicators.ForceIndex(Chart, Period, MAMethod,
AppliedPrice).Value.get(relativeShift + Shift);
    }
```

Sólo necesitamos llamar a Indicators. ForceIndex () con los parámetros correctos.

Esto es todo, ahora cuando presionamos Compilar y luego reiniciamos SQ veremos a nuestro nuevo indicador ForceIndex en la sección **Random Indicators Signals**.



Código fuente completo de nuestro nuevo indicador:

```
package SQ.Blocks.Indicators.ForceIndex;

import com.strategyquant.lib.*;
import com.strategyquant.datalib.*;
import com.strategyquant.tradinglib.*;

import SQ.Internal.IndicatorBlock;

@BuildingBlock(name="(FI) ForceIndex ", display="ForceIndex (#Period#)[#Shift#]",
returnType = ReturnTypes.Number)
@Help("ForceIndex help text")
public class ForceIndex extends IndicatorBlock {

    @Parameter
    public ChartData Chart;

    @Parameter(defaultValue="10", isPeriod = true, minValue=2, maxValue=1000, step=1)
    public int Period;

    @Parameter(name="Method", defaultValue="0")
    @Editor(type=Editors.Selection, values="Simple=0,Exponential=1,Smoothed=2,Linear
weighted=3")
    public int MAMethod;

    @Parameter(defaultValue="0")
    @Editor(type=Editors.Selection,
values="Close=0,Open=1,High=2,Low=3,Median=4,Typical=5,Weighted=6")
    public int AppliedPrice;

    @Output
    public DataSeries Value;

    //--------------------------------------------------------------------------
    //--------------------------------------------------------------------------
    //--------------------------------------------------------------------------

    @Override
    protected void OnBarUpdate() throws TradingException {
        double indiValue;

        indiValue = Chart.Volume.get(0) * (
           Indicators.MA(Chart.getSeries(AppliedPrice), Period, MAMethod).Value.get(0)
           - Indicators.MA(Chart.getSeries(AppliedPrice), Period, MAMethod).Value.get(1));

        Value.set(0, indiValue);
    }
    //--------------------------------------------------------------------------

    @Override
    public double OnBlockEvaluate(int relativeShift) throws TradingException {

        return Indicators.ForceIndex(Chart, Period, MAMethod,
AppliedPrice).Value.get(relativeShift + Shift);
    }
}
```

2.2 Creando una nueva señal basada en el nuevo indicador

Nuestro nuevo indicador **ForceIndex** está implementado, pero como se mencionó anteriormente, solo está disponible en la sección **Random Indicators Signals**. Esto significa que se combinará aleatoriamente con todas las comparaciones y valores numéricos aleatorios.

En StrategyQuant X deberías preferir el uso de señales, son condiciones predefinidas que usan un indicador, por ejemplo:

El Indicador está aumentando
El indicador está cayendo

El indicador cruza por encima del nivel.
El indicador cruza por debajo del nivel.

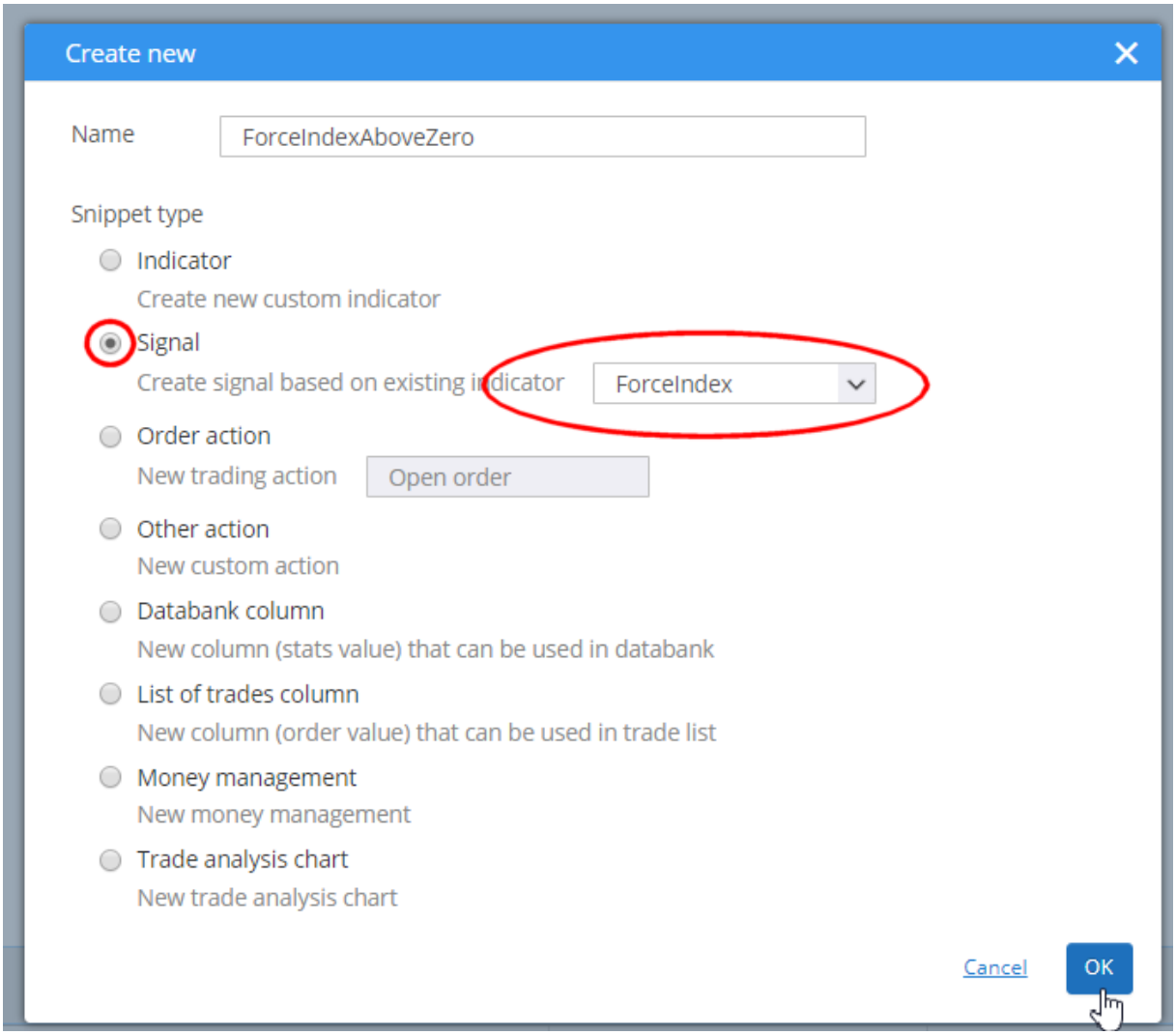
El uso de señales predefinidas como esta tiene muchas ventajas, una de ellas es que disminuye los niveles de libertad y le permite especificar condiciones que tienen cierto sentido en el trading. SQ utilizará estas condiciones predefinidas en lugar de intentar generar la condición completa al azar.

A continuación, te mostramos cómo puedes crear tus propias señales, que utilizan un indicador ya implementado – en nuestro caso ForceIndex. Para este ejemplo vamos a implementar dos señales sencillas:

ForceIndex es mayor que cero – debería sugerir una tendencia alcista
ForceIndex es menor que cero – debería sugerir una tendencia bajista

2.2.1 Paso 1 – Crear una nueva señal en Code Editor

Como indicador, Signal es un fragmento que debe ser programado en SQ. Abre CodeEditor, haz clic en crear nuevo y crea un nuevo fragmento con el nombre **ForceIndexAboveZero**. Elige **Señal** como tipo de fragmento, y **ForceIndex** como indicador en el que se basa la señal.



Esto creará un nuevo fragmento de código **ForceIndexAboveZero.java** y lo colocará en SQ -> Bloques -> Indicadores -> carpeta FoceIndex.

Cuando revisemos el código, notarás que el código es mucho más simple que el código del indicador. También tiene parámetros, que deben coincidir con los parámetros del indicador, y luego solo un método OnBlockEvaluate () que debe devolver verdadero o falso dependiendo de si la señal es válida o falsa.

Ten en cuenta que la plantilla de señales estándar podría no coincidir con tu indicador, como lo es en este caso. Si intentas compilarlo, recibirás errores de compilación, porque la llamada del indicador Strategy.Indicators.ForceIndex (Input, Period) en el método OnBlockEvaluate () no usa todos los parámetros requeridos.

En los siguientes pasos modificaremos la plantilla por defaukt para que funcione con ForceIndex

Implementar correctamente los parámetros de ForceIndex

Si observas el indicador ForceIndex notarás que tiene parámetros: Chart, Period, MAMethod, AppliedPrice.

Debemos copiar los mismos parámetros también a nuestra nueva señal:

```
public class ForceIndexAboveZero extends ConditionBlock {

    @Parameter
    public ChartData Chart;

    @Parameter(defaultValue="10", isPeriod = true, minValue=2, maxValue=1000, step=1)
    public int Period;

    @Parameter(name="Method", defaultValue="0")
    @Editor(type=Editors.Selection, values="Simple=0,Exponential=1,Smoothed=2,Linear
weighted=3")
    public int MAMethod;

    @Parameter(defaultValue="0")
    @Editor(type=Editors.Selection,
values="Close=0,Open=1,High=2,Low=3,Median=4,Typical=5,Weighted=6")
    public int AppliedPrice;
```

o, alternativamente, es posible que desees omitir algunos de los parámetros, y utilizar su valor fijo en su lugar. La creación de un parámetro significa que será editable en Wizard y SQ será capaz de generar valores aleatorios para este parámetro.

Hay un parámetro más que debemos añadir:

```
    @Parameter
    public int Shift;
```

En contraste con el fragmento de indicador, el parámetro Shift en las señales no se utiliza de forma predeterminada y debe agregarse explícitamente.

Implementar método OnBlockEvaluate

A continuación, se muestra una implementación del método OnBlockEvaluate () para nuestra señal. Como se puedes ver, es bastante simple-sólo tiene que obtener el indicador llamando a Strategy. Indicator. ForceIndex (...) con los parámetros adecuados, y luego simplemente obtener su valor en el desplazamiento especificado y compararlo con cero.

```
@Override
    public boolean OnBlockEvaluate() throws TradingException {
        ForceIndex indicator = Strategy.Indicators.ForceIndex(Chart, Period, MAMethod,
AppliedPrice);
        double value = indicator.Value.getRounded(Shift);

        return (value > 0);
    }
```

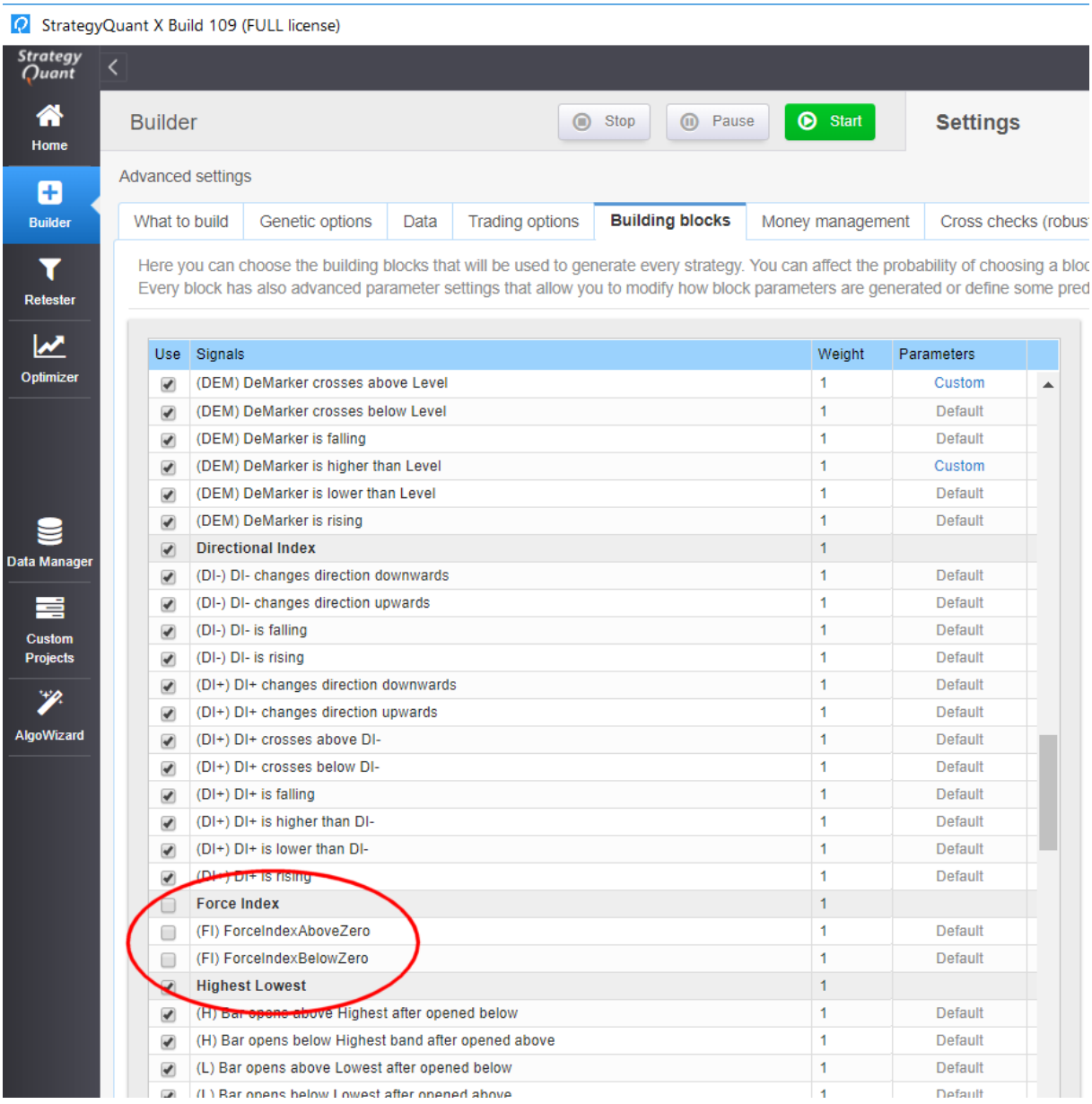
Esto es todo, nuestra nueva señal ya está terminada, podemos hacer clic en compilar y debería ser compilado con éxito.

Implementar ForceIndexBelowZero

De la misma manera debemos aplicar también **ForceIndexBellowZero. La única diferencia de la señal anterior es que la comparación en el método** OnBlockEvaluate () **será opuesto:**

```
return (value < 0);
```

Ahora compila también la segunda señal y luego reinicia SQ. Cuando veas los bloques de construcción deberías ver una nueva sección para ForceIndex en las señales, con nuestras dos nuevas señales:



Ten en cuenta que la funcionalidad de CodeEditor en SQ X está todavía en desarrollo. Estamos trabajando en hacerlo más documentado y más fácil de usar.
Si algo no funciona mientras tanto, por favor avísanos abriendo una solicitud de error o función en nuestro sistema de tareas.
Además, si tienes problemas para agregar un nuevo fragmento de código, podemos hacerlo por ti.

3. Valores estadísticos y columnas

3.1 Agregar nueva columna al banco de datos (valor estadístico)

Es posible que desees ampliar StrategyQuant mediante la adición de una nueva columna en el banco de datos, que compute algunas estadísticas únicas que son importantes para ti.

Cada valor estadístico como ganancia neta, factor de ganancia, drawdown, Sharpe ratio etc., se implementa como fragmento de columna en el Banco de datos. El fragmento de código se encarga de calcular el valor y devolverlo en el formato adecuado para que se muestre en la cuadrícula.

Usaremos términos estadísticos como: valor, valor estadístico, columna del banco de datos de una manera intercambiable, significan lo mismo.

Hay dos maneras básicas cómo el valor de las estadísticas puede ser calculados:

1. De trades
2. Como alguna relación entre otros valores estadísticos ya calculados

3.1.1 NetProfit-ejemplo de valor estadístico del cálculo de los trades

A continuación, se muestra un código para calcular el beneficio neto. Simplemente pasa a través de la lista de trades y calcula su PL.

```
    public double compute(SQStats stats, StatsTypeCombination combination, OrdersList
ordersList, SettingsMap settings, SQStats statsLong, SQStats statsShort) throws Exception {

        double netProfit = 0;

        for(int i = 0; i<ordersList.size(); i++) {
            Order order = ordersList.get(i);

            if(order.isBalanceOrder()) {
                // don't count balance orders (deposits, withdrawals) in
                continue;
            }

            double PL = getPLByStatsType(order, combination);

            netProfit += PL;
        }

        return round2(netProfit);
    }
```

La única "especialidad" aquí es el método auxiliar getPLByStatsType (order), que devuelve PL ya sea en dinero, pips o %.

3.1.2 Ret/DD Ratio – ejemplo de cálculo de valores estadísticos de otros valores

Algunos valores de estadísticas se calculan como proporciones de otros valores estadísticos. Un ejemplo de esto es la relación RET/DD, que es simplemente una relación entre la ganancia neta y el drawdown.

Aquí está su código:

```
    public ReturnDDRatio() {
        super(L.t("Ret/DD Ratio"), DatabankColumn.Decimal2, ValueTypes.Maximize, 0, -20,
20);

        setDependencies("NetProfit", "Drawdown");
        setTooltip(L.t("Return / Drawdown Ratio"));
    }

    //--------------------------------------------------------------------------

    @Override
    public double compute(SQStats stats, StatsTypeCombination combination, OrdersList
ordersList, SettingsMap settings, SQStats statsLong, SQStats statsShort) throws Exception {
        double netProfit  = stats.getDouble("NetProfit");
        double DD = Math.abs(stats.getDouble("Drawdown"));

        return round2(safeDivide(netProfit, DD));
    }
```

Llamando `setDependencies("NetProfit", "Drawdown");` en el constructor ReturnDDRatio le dijimos a StrategyQuant que este valor depende de otros dos valores de inicio que deben calcularse primero.

En el método compute () simplemente recuperamos estos valores y devolvimos su división.

3.2 Nueva columna Databank: relación entre valor principal y verificación cruzada avanzada

Basándonos en la solicitud del usuario, en este ejemplo agregaremos una nueva columna de base de datos que calculará la relación porcentual entre el retorno/drawdown de la estrategia original y el rendimiento de la prueba de robustez de Monte Carlo. En otras palabras, mostrará qué tan grande es el % de retorno/DD de la prueba de Monte Carlo del retorno/DD del resultado original.

Así que la fórmula será:

New value = {Return/DD of Monte Carlo cross check} / {Return/DD of backtest on main data}

Ten en cuenta que la funcionalidad CodeEditor en SQ X aún está en desarrollo. Estamos trabajando para hacerlo más documentado y más fácil de usar.

Si algo no funciona mientras tanto, avísanos abriendo una solicitud de error o función en nuestro sistema de tareas.

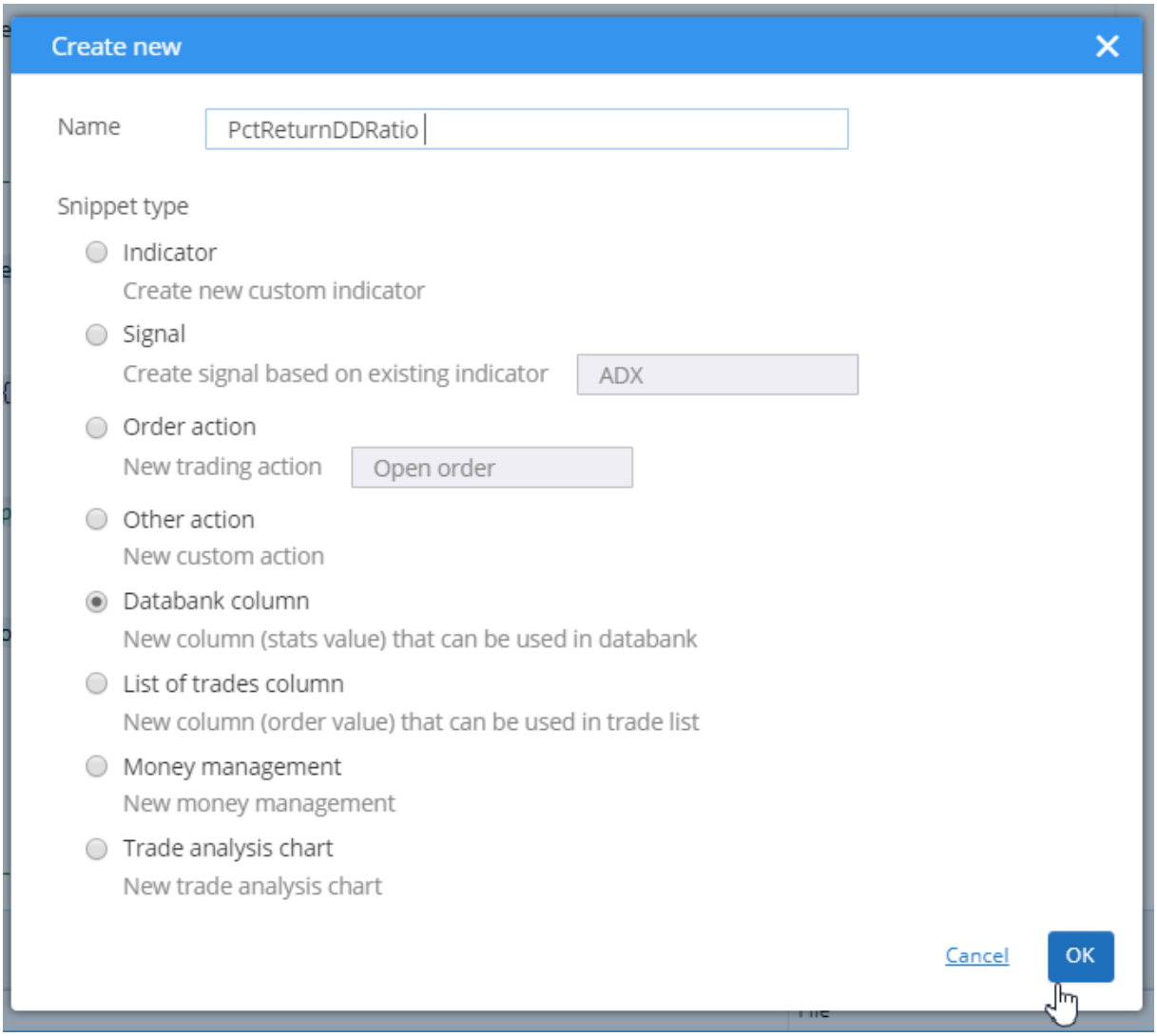
Además, si tienes problemas para agregar un nuevo fragmento de código, podemos hacerlo por ti.

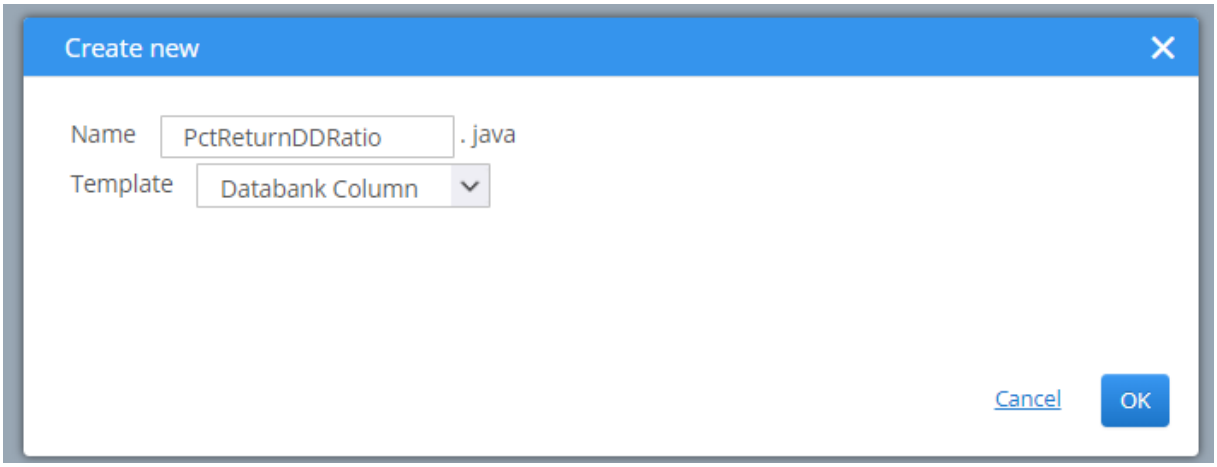
3.2.1 Paso 1 - Iniciar CodeEditor y crear un nuevo fragmento

Inicia el Editor de código usando el ícono de código en la parte superior derecha del programa.

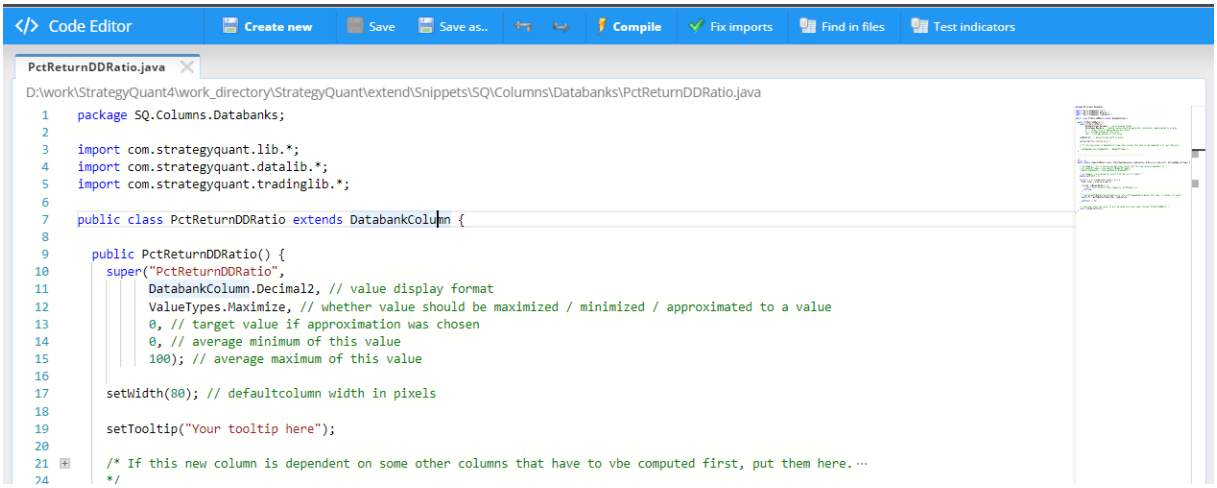
Como queremos agregar una nueva columna de base de datos, haremos clic en Crear nueva

Y elige la columna del Banco de datos:





Esto creará un nuevo fragmento de columna en el Banco de datos a partir de una plantilla estándar.



Este nuevo fragmento no hace mucho. Se agregará una nueva columna al banco de datos después de que vuelvas a compilar y reiniciar SQ, pero mostrará el valor de la ganancia neta normal porque este es el cálculo de ejemplo en la plantilla estándar.

3.2.2 **Paso 2 -** Agregar cálculo de valor al fragmento

Nuestro nuevo fragmento de código será especial – no calculará un nuevo valor estadístico de la lista de operaciones, sino que calculará una relación entre los valores ya existentes.

Así que tenemos que modificar el fragmento de este modo:

```
    public PctReturnDDRatio() {
        super("PctReturnDDRatio", DatabankColumn.Decimal2Pct, ValueTypes.Maximize, 0, 0,
100);

        setDependencies("ReturnDDRatio");
        setTooltip(L.t("Pct of Return / Drawdown Ratio between MC and main backtest"));
    }
```

El primer método es el constructor de fragmentos. Aquí podemos nombrar el fragmento de código, ponerlo en el valor de salida en% y que debe ser maximizado (mayor valor es mejor).

Debido a que usará el valor de ReturnDDRatio vamos a establecer la dependencia en él.

```
    @Override
    public double compute(SQStats stats, StatsTypeCombination combination, OrdersList
ordersList, SettingsMap settings, SQStats statsLong, SQStats statsShort) throws Exception {
        double mainReturnDD = stats.getDouble("ReturnDDRatio");

        return mainReturnDD;
    }
```

El segundo método se utiliza para calcular el valor de la lista de órdenes. Normalmente aquí es donde vamos a calcular y devolver el nuevo valor. Pero en nuestro caso no estamos comparando nuevo valor

de la lista de órdenes, estamos calculando la proporción de valores ReturnDDRatio ya existentes y comparados a partir de dos pruebas diferentes – Principal y verificación cruzada.

Así que en nuestro caso este método sólo obtiene y devuelve el valor para este campo.

El tercer método es donde ocurre el cálculo real. Es un método especial que fue creado especialmente para permitir las relaciones de cómputo entre los valores de verificación cruzada y de prueba principal.

```
    @Override
    public String getCrossCheckComputedValue(ResultsGroup results, double crossCheckValue)
throws Exception {

        // Return/DD of configured cross check
        double ccReturnDD = crossCheckValue;

        // Return/DD of main result
        double mainReturnDD = results.portfolio().stats(Directions.Both, PlTypes.Money,
SampleTypes.FullSample).getDouble(StatsConst.RETURN_DD_RATIO);

        double pct = SQUtils.safeDivide(ccReturnDD, mainReturnDD) * 100;

        return String.format("%.2f", pct);
    }
```

Este método hace:

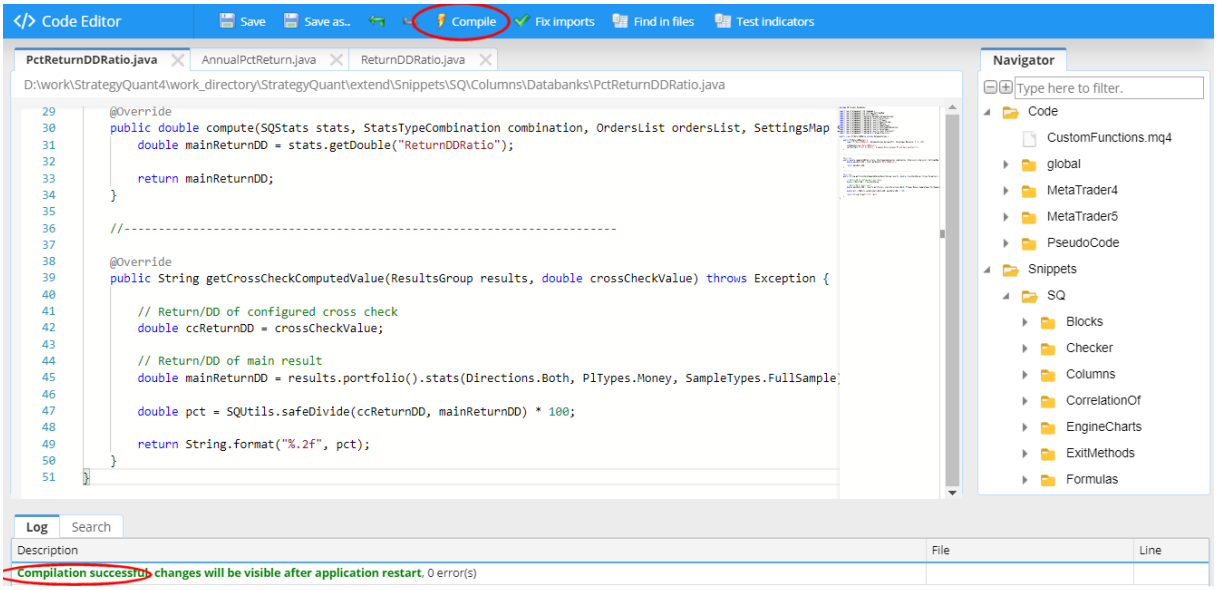
1. Obtener valor de Retorno/DD de esta verificación cruzada – se calculó antes en el método Compute () y se da como parámetro crossCheckValue

2. Obtener Retorno/DD del resultado principal – llamando Results. portafolio (). stats (directions. ambos, PlTypes. Money, SampleTypes. FullSample). GetDouble (StatsConst. RETURN_DD_RATIO)

3. Calcula el porcentaje de estos dos valores y lo devuelve

3.2.3 Paso 3 - Compile los fragmentos y reinicie SQ

Una vez que haya terminado, puedes presionar el botón Compilar en la barra de herramientas superior. Si no hubo ningún error, la compilación tendrá éxito y tu nueva columna será visible la próxima vez que reinicies SQ.



Podría haber algunos errores de compilación, algunos de ellos podrían ser causados por las importaciones perdidas.
Al hacer clic en corregir importaciones debes resolver esto. Si todavía ves algunos errores puede ser un error en el código, o algo causado por el editor de código no funciona correctamente.

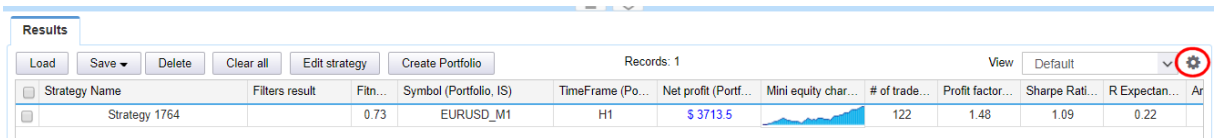
Ten en cuenta que la funcionalidad CodeEditor en SQ X aún está en desarrollo. Estamos trabajando para hacerlo más documentado y más fácil de usar.

Si algo no funciona mientras tanto, avísanos abriendo una solicitud de error o función en nuestro sistema de tareas.

Además, si tienes problemas para agregar un nuevo fragmento de código, podemos hacerlo por ti.

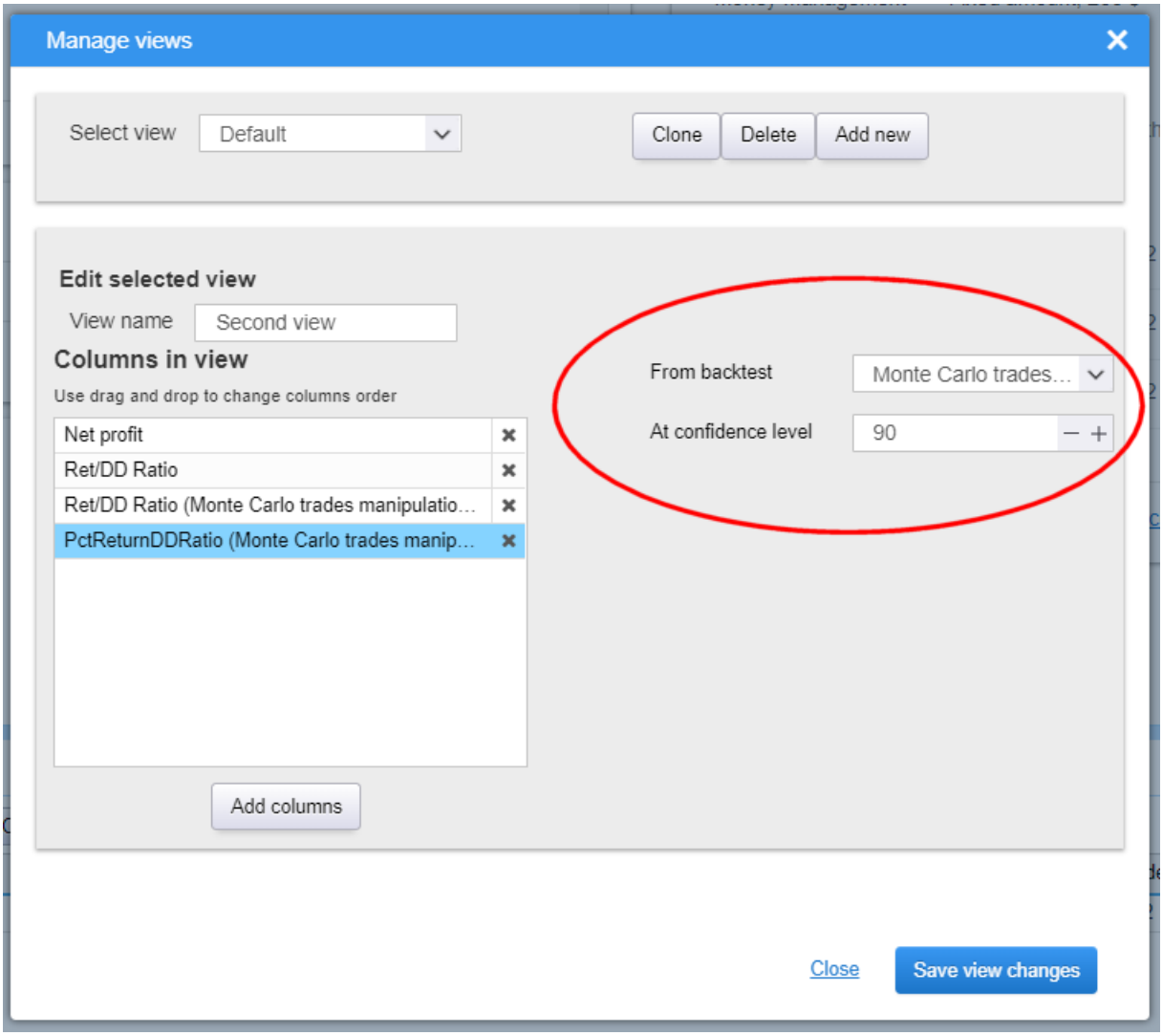
3.2.4 Paso 4 – Usar la nueva columna del Banco de datos

Cuando la compilación fue exitosa puedes reiniciar SQ. A continuación, ve a Administrar vista para utilizar la columna recién creada.

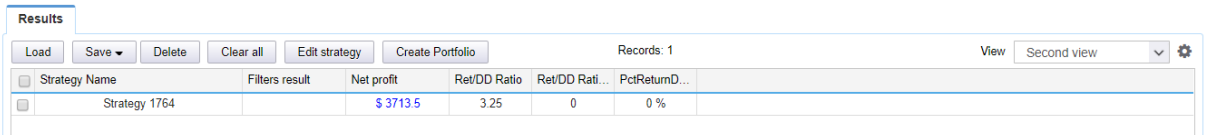


En la vista Administrar, haz clic o Agregar nueva vista, nombra por ejemplo **Segunda vista** y agregale algunas columnas.

Como una de las columnas vamos a añadir también nuestra nueva columna **PctReturnDDRatio**
Nuestra nueva columna debe estar correctamente configurada. Queremos comparar el valor de la verificación cruzada con el valor del backtest principal. Así que tenemos que configurarlo para calcular tus datos de Monte Carlo verificación cruzada con nuestro nivel de confianza deseado.
Este será el valor que será devuelto por el método Compute () y que vendrá como un crossCheckValue parámetro en el getCrossCheckComputedValue ().

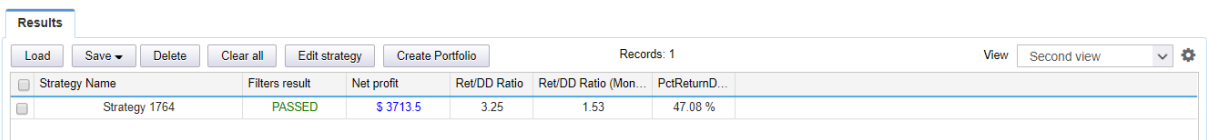


Cuando hayamos terminado, podemos hacer clic en Guardar y se agregará nuestra nueva vista. Cuando cambiamos el banco de datos a una nueva vista, vemos que hay nuevas columnas, pero tienen valor 0.



Esto se debe a que la estrategia en el banco de datos no se probó con la verificación cruzada de Monte Carlo y, por lo tanto, todavía no contiene ningún resultado de Monte Carlo.

Cuando volvamos a probar la estrategia con esta verificación cruzada veremos los datos:



Retorno/DD prueba principal es de 3.25, Return/DD verificación cruzada de Monte Carlo es 1.53, y su relación de porcentaje (nuestra nueva columna **PctReturnDDRatio**) se calculó a 47.08 % que es un valor correcto.

También puede descargar el fragmento de este ejemplo aquí:
https://strategyquant.com/downloads/PctReturnDDRatio.java

¡ Ten en cuenta!

Vamos a añadir más ejemplos de fragmentos para StrategyQuant X así como documentación más detallada, por favor ten paciencia.

Si tienes una solicitud específica, por favor envíala como una solicitud de función en nuestro sistema de tareas y podríamos implementarla por ti.