IM Trust - Credicorp Capital
Administradora General de Fondos
Mesa de Gestión de Inversiones
Diego Steinsapir

1 de febrero de 2017

Generación de Excel para Reporte de Seguimiento

1. Introducción

Para cada cartera, es necesario periodicamente producir un Reporte de Seguimiento en el cual se especifique la evolución de su rendimiento. De esta manera, el Reporte de Seguimiento muestra los cambios en la rentabilidad de la cartera y su composición para un periodo, además de los movimientos generados y junto con más información relevante. En el presente documento se detallan las principales características, usos y explicaciones del programa y sus principales métodos creado para la generación automática del archivo Excel utilizado para los Reportes de Seguimiento.

2. Generación de Reporte de Seguimiento

2.1. Materiales

Para la generación automática del archivo Excel con la información necesaria para el Reporte de Seguimiento, es necesario contar con dos archivos:

- generador.exe: Archivo ejecutable encargado de recopilar y analizar datos necesarios para transformarlos en información utilizable en el Reporte de Seguimiento
- seguimiento.xlsm: Archivo de Excel preformateado sobre el cual seguimiento.exe lee paramétros iniciales y luego edita con la información final para Reporte.

2.2. Modo de Uso

Para generar el Excel para el Reporte de Seguimiento de manera automática, se deben seguir los pasos a continuación:

1. Abrir seguimiento.xlsm y en la hoja Çontrol.^{es}cribir el código del fondo a evaluar y las fechas entre las cuales se desea evaluar. Las fechas deben ir en formato 'AAAA-MM-DD'. NO editar nada más.Guardar y cerrar archivo

2. Ejecutar generador.exe. Elegir archivo seguimiento.xlsm editado cuando aparezca el diálogo pidiéndolo. Esperar a que se termine de ejecutar cuidando no interactuar con el Excel que se abra.

2.3. Output

Tras finalizar la ejecución, el programa generará un archivo Excel con todas las tablas y gráficos en formatos correctos listos para su utilización en el Reporte. Dicho programa se generará en una carpeta a eleccin por el usuario y tendrá el nombre CodigoFondo_FechaCreación.xlsm

	A	B
1		
2	Cliente/fondo	INV_MALLIN
3	Fecha Inicio	27-01-2014
4	Fecha Fin	31-10-2016
5	Serie	
6	Lista	Si
7	Fijar Bmk	

Figura 1: Grafico 1 Ejemplo de edición de hoja Control

3. Detalle de Archivos

A continuación, se describirá de manera más específica cada archivo utilizado.

3.1. seguimiento.xlsm

3.1.1. Estructura

El archivo seguimiento.xlsm corresponde a un libro de Excel con Macros habilitados. En el se poblarán los datos relevantes para el Reporte de Seguimiento y se generarán las tablas y gráficos necesarios. Se entrega con una estructura establecida y con el diseño de tablas y gráficos corporativo incluido.
Se estructuran las hojas de la siguiente manera:

- Control: En esta hoja se deben ingresar los parámetros a leer para la generación del resto del archivo Excel
- Datos: Aquí se imprimirán los valores cuota del portafolio, junto con retornos mensuales y anuales para cada uno. Los retornos se calculan con el programa generador.exe.
- SEGUIMIENTO: Aquí se imprimirán las tablas y gráficos relevantes para el Reporte de Gestión con su formato y diseño correcto. Es importante destacar que estos elementos se generan desde el Excel mismo. Vale decir, el archivo seguimiento.xlsm viene con los gráficos creados, pero sin los datos neceasrios. Esto se hace de esta manera para permitir la edición de datos si es necesario, además de dar una mayor visibilidad del origen de los datos mismos. Finalmente, ello permite asegurar de antes (y editar a posteriori) el diseño utilizado en cada uno.
- CarteraCompleta: Se imprimen todos los instrumentos incluidos en el portafolio junto con todas sus clasificaciones de todo tipo.
- Cartera-Neutral: Se imprime el detalle de la Cartera-Neutral [1]
- CarteraRFL, CarteraRVL, CarteraRFI, CarteraRVI: Se imprimen los instrumentos de cada clase de activo mayor (RFL, RVL, RFI, RVI), separados por subclase
- ResumenCartera: Se imprimen las tablas de las principales descomposiciones de la cartera completa en monto y pocentaje. Por ejemplo:

Moneda	Monto	Porcentaje
$	10.000	0,25
UF	30.000	0,75

[1] 29-12-2016: Aun debe hacerse, pero hoja está creada

- ResumenCarteraLabel: Igual a hoja ResumenCartera", pero quita columnas de montos y agrega etiquetas a entradas para generar gráficos con formato correcto
- ResumenCarteraRF: Igual a ResumenCartera, pero creado de forma específica para la Renta Fija Local
- ResumenCarteraRFLabel: Equivalente a ResumenCarteraLabel pero para Renta Fija
- Información Cartera/Argupar Información Cartera: Botón para esconder/mostrar pestañas de desgloses de cartera para hacer Excel más ordenado
- Evolucion: En esta hoja estarán disponibles los gráficos de la evolución de la cartera según su composición por clase de activo, moneda y mercado
- DataEvolucion: Aquí se encontrará en forma de tabla la información necesaria para generar los gráficos de la evolución de la cartera en la hoja .Evolucion"

3.1.2. Avance futuro

Se debe agregar en el futuro una hoja en que se detallen los movimientos de la cartera, como aportes y rescates y pagos de dividendos

3.2. generador.exe

En esta sección se explican los métodos más importantes y complejos del código junto con el flujo completo y desiciones de diseño y tecnologías.

3.2.1. Tecnología

El programa fue escrito con Python 3.5.2 y está versionado con Git en el repositorio

`mesagi/Proyectos/generador_presentacion_gestionactivos`

3.2.2. Diseño

Para mayor mantenibilidad y facilidad de lectura, el código se ha separado en distintos archivos. Un archivo llamado generador.py contiene el flujo principal del program (método "main"), otro llamado seguimiento.py es la librería con los métodos utilizados por generador.py, y el resto de los archivos corresponde a las consultas SQL necesarias para el programa.
Dentro del código se prioriza no usar variables globales (no así, constantes globales) para faciilitar el entendimiento del código. Al principio de cada archivo se especifican las constantes utilizadas (nombres de hojas, parametros y entradas en bases de datos). Si por alguna razón alguno de estos valores se cambiará, basta cambiar los valores al principio del archivo para volver al funcionamiento normal.

3.2.3. Principales Métodos

- calculateReturns(data, index): Calcula los retornos mensuales y anuales de un portafolio y su benchmark a partir de los valores diarios.

```
def calculateReturns(data, index):  # 1 para fondo, 2 para benchmark
    currentDayForMonth = data[0] # Init values
    currentDayForYear = data[0]
    monthlyReturns = []
    yearlyReturns = []

    # NOTE Datos ya vienen ordenados en forma desc
```

```
for day in data:
  if currentDayForMonth[0].month != day[0].month:
      currentMonthReturn = currentDayForMonth[index] /
                           day[index] - 1
      currentMonthBenchmarkReturn = currentDayForMonth[index + 1] /
                                    day[index + 1] - 1
      monthlyReturns.append([str(currentDayForMonth[0].month)
                             + "/" + str(currentDayForMonth[0].year),
                             currentMonthReturn,
                             currentMonthBenchmarkReturn])
      currentDayForMonth = day
      if currentDayForYear[0].year != day[0].year:
          currentYearReturn = currentDayForYear[index] / day[index] - 1
          currentYearBenchmarkReturn = currentDayForYear[index + 1] /
                                       day[index + 1] - 1
          yearlyReturns.append([currentDayForYear[0].year,
                                currentYearReturn,
                                currentYearBenchmarkReturn])
          currentDayForYear = day
```

En esta primera parte, se aprovecha que los valores de la cartera y el benchmark vienen ya ordenados por fecha. Luego, se toma el primer valor de estos datos y se itera sobre las demás entradas hasta deetectar un cambio de mes. Una vez que se identifica un cambio de mes, se divide el valor original por el primero del mes nuevo (dia 31 o 30 del mes anterior) para obtener la rentabilidad de ese mes. Luego, se actualiza el valor contra el cual comparar y se hace lo mismo para el siguiente mes hasta terminar los datos. Cuando se detecta un cambio de mes se verifica también un cambio de año para obtener el retorno anual con la misma lógica.

Lo anterior deja afuera la rentabilidad del último mes y año (más lejano al día de hoy). Para incluirlo, el método termina de la siguiente manera:

```
# Incluir ultimo mes
last = data[len(data) - 1]
if currentDayForMonth[0].month == last[0].month:
  lastMonthReturn = currentDayForMonth[index] / last[index] - 1
  lastMonthBenchmarkReturn = currentDayForMonth[index + 1] /
                             last[index + 1] - 1
  monthlyReturns.append([str(last[0].month) + "/" + str(last[0].year),
                         lastMonthReturn, lastMonthBenchmarkReturn])

# Incluir ultimo anio
if currentDayForYear[0].year == last[0].year:
  lastYearReturn = currentDayForYear[index] / last[index] - 1
  lastYearBenchmarkReturn = currentDayForYear[index + 1] /
                            last[index + 1] - 1
  yearlyReturns.append([last[0].year,
                        lastYearReturn,
                        lastYearBenchmarkReturn])

return (monthlyReturns, yearlyReturns)
```

- subclassify(portfolio): Se utiliza para entregar a cada instrumento en el portafolio las subcategorías que son relevantes para el Reporte de Seguimiento pero que no están explícitas en las bases de datos

```
def subclassify(portfolio):
    # Indices de columnas
    instrumentTypeIndex = portfolio[0].index(TIPOINSTRUMENTO)
    zoneIndex = portfolio[0].index(ZONA)
    durationIndex = portfolio[0].index(DURATIONHEADER)
    riskIndex = portfolio[0].index(CLASIFICACIONRIESGO)
    currencyIndex = portfolio[0].index(MONEDA)
    incomeTypeIndex = portfolio[0].index(RENTA)
    amountIndex = portfolio[0].index(MONTO)
```

Primero se busca en qué indice dentro del arreglo está cada título de columna de la tabla. Cabe notar que portafolio es una tupla de dos datos, en que el primer datos corresponde a un arreglo de títulos y el segundo a los datos mismos. Esto deja a portfolio de la forma portfolio = ¡Titulos, Datos¿. También es importante destacar que las constantes que se están dando como parámetro en cada caso corresponde a los títulos actuales en la base de datos (por ejemplo, MONEDA = "Moneda"), que en caso de cambiar deben modificarse también en este archivo al principio. Esta lógica se utiliza en más métodos.

```
i = 0
totalAmount = sum(row[amountIndex] for row in portfolio[1])
for instr in portfolio[1]:
        # Porcentaje. NOTE: Float necesario por bug xlwing
        percentage = float(instr[amountIndex] / totalAmount)
        # Zona
        zone = instr[zoneIndex]
        # Corto_Largo
        short_long = 'Corto' if instr[durationIndex] <= DURATION else 'Largo'
        # Moneda
        currency = instr[currencyIndex]


                                    ...
```

Esto se continua para varias mas categorías. Se agrega ademas lo siguiente para poder detectar los fondos que serian subclasificados como liquidez y como mal clasificados. Aquellos mlas clasificados se les entrega la asset_class y subclase virtual 'NC' (No Clasificado).

```
        # Asset Class y Subclass, tomando en consideracion que podrian ser liquidez
        if "CFM8401" in instr[instrumentCodeIndex] or
          instr[emmitterIndex] in ["LIQUIDEZ", "CAJA"]:
            assetClass = "Liquidez " + instr[currencyIndex]
            subclass = "Liquidez " + instr[currencyIndex]
        else:
            assetClass = getClass(incomeType, zone, country)
            subclass = getSubclass(zone, currency, risk,
                       instrumentType, short_long, incomeType, country)

        # Clasificacion Falsa para detectar incorrectos
        if detectWrongClassification(instr):
            assetClass = "NC"
            subclass = "NC"
```

Luego, se agrega en el arreglo de datos todas las nuevas categorías:

```
        # Extender todo
        instr += (percentage,
                  short_long,
                  risk_duration_currency,
                  risk_duration,
                  duration_currency,
                  instrumentType_Currency,
                  instrumentType_duration,
                  assetClass,
                  subclass)

        portfolio[1][i] = instr
        i += 1
```

Con el loop finalizado, se agregan los nuevos títulos y se retorna portfolio. El método entonces solo ha extendido nuevas categorías a los datos recopilados desde la base de datos:

```
    # Agregar nuevos titulos
    portfolio[0].extend(["Porcentaje",
                         CORTOLARGO,
                         RIESGODURACIONMONEDA,
                         RIESGODURACION,
                         DURACIONMONEDA,
                         TIPOINSTRUMENTOMONEDA,
                         TIPOINSTRUMENTOLARGO,
                         ASSETCLASS,
                         SUBCLASS])

    return portfolio
```

- assetClassBreakdown(headers, subportfolio): Divide un portafolio por subclases y lo devuelve como un diccionario con cada subclase como llave y los instrumentos de esa subclase en un arreglo como valor de la llave. La siguiente linea se encarga de obtener un arreglo con todas las llaves del diccionario de forma única a partir de las subclases observadas en el subportafolio:

  ```
  subclasses = set(np.column_stack(subportfolio)[subclassIndex])
  ```

  Después de esto, se genera un diccionario y se itera para agregarlos al arreglo de instrumentos correspondiente a su subclase en la llave del diccionario.

- totalPerType(typeIndex, amountIndex, matrix): Se utiliza para la generación de los resumenes de cartera y devuelve el monto y porcentaje invertido en los instrumentos según alguna categoría. Se le entrega un indice de la columna de la cual se quiere hacer resumen en typeIndex y el índice en el que se encuentra el monto invertido en ese instrumento, junto con la matriz de datos. Por ejemplo, si se tiene una matriz de datos en que las columnas siguen el orden ¡Moneda, TipoInstrumento, Emisor, Riesgo, Monto, Duracion¿y se quiere hacer un resumen sobre los Emisores, el método se llama como:

  ```
  totales = totalPerType(2, 4, data)
  >>> totales = [
  >>> Emisor1: [15.000, 0,2],
  >>> Emisor2: [30.000, 0,4],
  >>> Emisor3: [30.000, 0,4]]
  ```

- printPortfolioSummary(workbook, sheet, portfolio, RFL): Utiliza el método totalPerType para imprimir en Excel todos los resumenes que resultan interesantes. Se agrega un valor booleano RFL para distinguir las categorías que se quieren resumir en caso que se esté evaluando la subcartera de Renta Fija Local

- printLabeledSummary(workbook, sheet, totals): Imprime lo mismo que el método anterior, pero con la diferencia que agrega etiquetas con formato listo para generar gráficos en Excel.

- printPortfolioCompositionPies(workbook, sheet, datasourceSheet, piesAmount, lastIndex): Se utiliza para generar los gráficos de torta. Es necesario destacar que en el Excel hay otros gráficos que se arman directamente desde el mismo archivo, pero los gráficos de torta no se comportaban muy bien si se dejaban prearmados con los datos listos. Para solucionar esto, se crean en el archivo Excel gráficos con el diseño correcto, pero sin datos. Luego, el programa lee cada uno de estos objetos y los llena con los datos correctos. El método funciona primero leyendo los objetos çharts"de una determinada hoja y luego asignándole su conjunto de datos.

- getStatistics(data): Obtiene estadísticas importantes para el set de datos de valores cuota de portafolio y benchmark entregados. Cada dato de la matriz es de la forma [Fecha, Retorno Portafolio, Retorno Benchmark]. Los indicadores estadísticos que entrega son:

  - Alpha
  - Beta
  - Tracking Error
  - R cuadrado
  - Volatilidad Portafolio
  - Volatilidad Benchmark

  La obtención de todas las estadísticas - a excepción del Tracking Error - se basa en el código encontrado en http://gouthamanbalaraman.com/blog/calculating-stock-beta.html, donde además hay una explicación de cada una. El tracking error por su parte, se calculó como la varianza del vector de alpha's, con alpha calculado como la diferencia entre los retornos de la cartera y el benchmark. Vale decir:

$$\text{Tracking Error} = \text{var}\begin{pmatrix}\alpha_1\\ \alpha_2\\ \vdots\\ \alpha_N\end{pmatrix} = \text{var}\begin{pmatrix}R_1 - R_1^b\\ R_2 - R_2^b\\ \vdots\\ R_N - R_N^b\end{pmatrix}$$

- extractByTimedelta: De un conjunto de datos, retorna aquellos que se encuentren dentro de una diferencia (delta) de fechas determinada.

- binarySearchFirstEntry: Implementación del algoritmo de búsqueda binaria ($\Theta(\log(n))$, $O(n)$) para encontrar la fecha del primer dato disponible dentro del rango de un rango de fechas determinado. Por ejemplo, si se pregunta por datos de una cartera entre 05-11-2010 y 21-05-2016 y el primer dato corresponde al 12-09-2014, este método retorna 12-09-2014.

  Nota: Se optó por esto debido a la complejidad de la base de datos. Buscar según MIN(fecha) en la tabla ZHIS_CARTERAS_MAIN demora mucho, mientras la consulta usada para este método era mucho menos costosa y puede usarse varias veces debido a esto.

- getLastPortfolioOfMonth: Se utiliza para encontrar la composición del último portafolio registrado de un mes. Se le entrega una fecha correspondiente al último día de un mes (ej: 31-03-2015) y la consulta que devuelve los instrumentos encotnrados en el portafolio a esa fecha. Luego, va iterando hacia atras hasta encontrar un día del cual se cuenten con registros.

- portfolioCompositionTimeSeries: Utilizado para encontrar la evolución de la cartera en el tiempo en tériminos de su composición por clase de activo, moneda y mercado en el que se invierte. Primero utiliza binarySearchFirstEntry para encontrar la fecha del primer registro. Con este dato, encuentra todas las fechas desde esta fecha hasta la fecha consultada y las remuestrea para tener solamente el último día de cada mes. Tras esto, las deja en formato correcto. To ello se hace en las siguientes líneas:

```
for i in range(1, delta.days + 1):
    dates.append(minDate + td(days=i))

samples = pd.DataFrame(dates, columns=["Date"])
samples = samples.set_index("Date")
samples = samples.resample('M').last()
samples.reset_index(drop=False, inplace=True)
samples.drop(0, inplace=True)
```

- printPortfolioEvolution: Este método se utiliza para poder imprimir la evolución de la cartera en términos de clases de activo, moneda y mercado de inversión. Se utiliza de la mano de portfolioCompositionTimeSeries. En particular, el método comienza por dejar la información de la evolución de la cartera en forma matricial para lograr un manejo de los datos más sencillo. Luego, lee los objectos 'Charts' ya creado en el Excel en la página 'evolucion' y les asigna su dataset para que se cree el gráfico correspondiente. Es equivalente a la creación de los gráficos de pies.

- main: El programa principal es bastante directo. Primero genera una conexión a la base de datos con las constantes declaradas al principio del programa. Luego hace una consulta a la base de datos por los valores cuota diarios entre las fechas especificadas para el portafolio y su benchmark, desde lo cual calcula los retornos. Luego consulta por los instrumentos que componen el portafolio y los imprime primero de manera completa, después de manera desglosada, y finalmente de manera resumida.

- resource_path: Sirve para leer archivos tanto en modo de desarrollo (corriendo el código desde la terminal sin un ejecutable) y en modo de producción (desde el ejecutable compilado). Este método es necesario para poder distribuir el archivo ejectuable sin necesidad de además distribuir las queries a la base de datos. El uso de MEIPASS, se explica en la documentación de PyInstaller en este link

3.2.4. Compilación

La compilación de generador.exe se hizo utilizando la librería PyInstaller Versión 3.2, con un equipo con Windows 10 instalado (64 bits) y Python 3.5.2, además de todas las librerías neceasrias para el programa. La compilación puede, en teoría, lograrse con otras librerías creadas con este fin (Py2Exe, cx_Freeze, etc.), sin embargo en este caso solo se logró con PyInstaller. Para el detalle de la compilación paso por paso, referirse al documento Manual de Compilación

3.2.5. Posibles extensiones

- Los errores debiesen manejarse de mejor manera, mostrando los errores nativos de Python además del mensaje de error que hay actualmente.

4. Puntos de Falla

- Existe un bug que ocurre a veces al finalizar la ejecución, en que Excel se congela o no corre bien. Basta cerrar todas sus instancias y volver a correr Excel. Se puede arreglar si se cambia la línea

```
workbook = Mesa.openWorkbook(filePath, False, False)
```

por

```
workbook = Mesa.openWorkbook(filePath, True, True)
```

Sin embargo, hace al programa mucho más lento. **RECOMENDACIÓN**: Guardar todo antes de correr el programa.

- A veces el programa se congela si hay alguna celda seleccionada y su formula en edición. Deseleccionar y debiese correr sin problemas