**# Diario**

**## Funcionamiento a vista de pájaro. Archivo [generar\_diario.py](**generar\_diario.py**)**

1. En el archivo [generar\_diario.py](generar\_diario.py) se ejecuta la función `main()`

   1. Se llama a la función `parse\_config()` que lee el archivo [config.yaml](config.yaml), donde hay algunos parámetros útiles para detectar errores cuando se modifique el código.

   2. Se llama a la función `\_get\_diary\_type\_from\_hour()` que basado en la hora de generación del diario devuelve el "opening" o el "closing"

   3. Se llama a la función `download\_data()`. Esta función descargará los datos correspondientes al día de hoy a través de la API de Eikon/Refinitiv/Reuters (o como quiera que se llame). **\*\*Esta función solo funcionará si estamos ejecutando el script desde la máquina virtual\*\***, ya que los ordenadores locales no tienen instalada la aplicación de escritorio de Eikon. Esta función llama a su vez al archivo [descargar\_datos\_eikon.py](descargar\_datos\_eikon.py) donde se realiza toda la descarga de datos (**\*\****\*ver sección siguiente\****\*\***)

   4. Se crea el documento word donde se escribirán las tablas, los gráficos y el texto, llamado [template\_diario.docx](template\_diario.docx)

   5. Se llama a la función `styles()` que formatea la sección donde se encuentran el título y subtítulo del documento word

   6. Se llama a la función `fill()` que rellena el documento Word (utilizando python-docx) con un grupo estructurado de tablas, títulos, gráficos y parágrafos basado en las definiciones de los mismos hechos en los archivos pertienentes. Esta función llama a su vez a las siguientes funciones:

      \* Función `bolsas\_plots()` para generar un doble gráfico donde se muestre la evolución de los principales índices de EEUU y de Europa

      \* Función `generate()` dentro del archivo [generar\_tablas\_word.py](generar\_tablas\_word.py) que, a su vez llamando a múltiples otras funciones, se ocupa de generar la estructura de las tablas (**\*\****\*ver sección correspondiente más abajo\****\*\***)

      \* Función `divisas\_plots\_doble()` que se ocupa de crear un doble gráfico, uno para las divisas y otro para el bitcoin y metales preciosos

      \* Función `materias\_primas\_plots()` para generar un gráfico con la evolución temporal de las distintas materias primas.

   7. Se llama a la función `footer()` que a su vez llama a la función `\_legend()` para añadir al documento Word (utilizando python-docx) los pies de página y la leyenda especificada.

   8. Se llama a la función `save()` que, partiendo del documento [template\_diario.docx](template\_diario.docx), formatean el documento final a partir de los datos descargados.

**## Funcionamiento de la descarga de datos: añadir el dato diario a la serie vía Reuters mediante el archivo [descargar\_datos\_eikon.py](**descargar\_datos\_eikon.py**)**

1. El histórico de datos se almacena en la carpeta [historic](historic/). Esta contiene archivos con extensión [.feather](https://arrow.apache.org/docs/python/feather.html), en la práctica, lo mismo que archivos con extensión .csv, pero no aptos para la lectura humana, a cambio, ocupan menos espacio en disco y el ordenador es capaz de leerlos más rápido.

   \* El archivo `historic\_{fecha}.feather` contiene datos de cierre de mercado desde el 31/12/2007 hasta el día laborable inmediatamente anterior a `fecha`.

2. De la descarga de archivos se encarga la función `generate\_diary\_tables()`, que está en el archivo [descargar\_datos\_eikon.py](descargar\_datos\_eikon.py).

   1. Carga en memoria el archivo que contiene datos hasta el antepenúltimo día laborable (el archivo `historic\_{hoy - 1}.feather` contiene los datos de cierre hasta "hoy-2" - anteayer - incluido).

   2. Se llama a la función `download\_new\_data()` dos veces para descargar:

      2.1. Los datos correspondientes al cierre del último día laborable (que entendemos consolidados), los añade al histórico cargado en memoria (que era `historic\_{hoy - 1}.feather` con datos de cierre hasta anteayer) y guarda el nuevo histórico en un archivo `historic\_{hoy}.feather` (con datos de cierre de mercado hasta ayer incluido)

      2.2. Los datos del día corriente (los datos correspondientes a la tarde del día de hoy cuando se elabora el diario), los añade al histórico cargado en memoria (`historic\_{hoy}.feather` con los datos de cierre de mercado hasta "hoy-1") y guarda el resultado en el archivo consolidado [flash.feather](historic/flash.feather), que contiene por tanto los datos de cierre de mercado hasta "hoy-1" (o sea, ayer) más el dato de la tarde que se produce el diario.

      2.3. Se llama a la función `\_fix\_new\_data()` que realiza una limpieza e imputación de datos sobre un DataFrame ("new\_data") rellenando valores faltantes o incompletos utilizando datos de conjuntos anteriores ("last\_data", "yield\_data")

   3. A partir de un dataframe ("updated\_data") elabora dos tablas de forma dinámica llamando a la función `\_generate\_table()`: una tabla para desviaciones (datos parecido a lo que sería un z-score y que se calculan llamando a la función `\_compute\_deviations()`) y la tabla de valores principales. Los archivos se guardan en archivos .csv y .feather. Se llaman a los diccionarios que respectivamente se encuentran en los archivos [tables.py](tables.py) y [rics.py](rics.py) para ir generando las tablas a partir de esta información.

   4. Para añadir a la serie los nuevos datos descargados y los cálculos ("new\_row") se llama a la función `add\_row()` que se encuentra dentro del archivo [historical\_deviation.py](historical\_deviation.py). Además maneja datos faltantes, calcula valores derivados, calcula diferencias con respecto a datos anteriores y actualiza estadísticas móviles y acumuladas, para lo cual llama a distintas funciones dentro del archivo [historical\_deviation.py](historical\_deviation.py) (**\*\****\*ver sección siguiente\****\*\***)

**## Funcionamiento de la manipulación y cálculo de datos: archivo [historical\_deviation.py](**historical\_deviation.py**)**

Este archivo se ocupa de los cálculos y manipulación de los datos, mediante los cuales se actualizan las tablas del diario Word. El archivo [historical\_deviation.py](historical\_deviation.py) es llamado por los siguientes dos archivos:

1. Archivo [reform.py](reform.py):

      Flujo de llamada de funciones: `reform()` ➝ `download\_full\_timeseries()` ➝ `process\_historic()` ➝ `process\_raw\_data()` ∈ [historical\_deviation.py](historical\_deviation.py)

      El flujo en [historical\_deviation.py](historical\_deviation.py) donde la llamada se produce desde el archivo [reform.py](reform.py) sería el siguiente:

      1. La primera función que se llama es `process\_raw\_data()` que contiene un DataFrame, "raw data", donde cada columna del mismo es procesada por la función `\_process\_raw\_data()`

      2. La función `\_process\_raw\_data()` toma una serie de valores (formato DateFrame) y la transforma en un formato más detallado a través de una serie Multiíndice (DataFrame) que incluye: (1) los valores originales, (2) las diferencias entre el valor actual y los valores pasados (tanto cambios absolutos como porcentuales) y (3) el promedio y la variación dentro de una ventana móvil de 20 años (media y varianza rodante). Esta función llama a su vez a dos funciones adicionales:

         \* Función `\_compute\_offset\_dates()` que se utiliza para definir puntos de referencia en el tiempo con el fin de comparar valores de una serie temporal (por ejemplo, para ver cómo se compara un valor actual con el día hábil anterior (ayer), el último viernes, una fecha fija relacionada con una crisis, etc.)

         \* Función `\_compute\_differences()` calcula las diferencias entre los valores originales de la serie temporal y sus valores correspondientes en varias fechas desplazadas, y luego almacena estas diferencias en una Serie con Multiíndice ya preexistente ("processed\_multicol"). Para ello, esta función llama a su vez a:

            \* Función `\_compute\_difference()`: calcula dos tipos de diferencias entre una serie temporal (original) y la misma serie desplazada a diferentes fechas - "offset dates" - (diff\_dates) y calcula tanto la diferencia absoluta (la diferencia simple entre los valores) como la diferencia relativa (la diferencia en relación con el valor desplazado (esencialmente un cambio porcentual)).

2. Archivo [descargar\_datos\_eikon.py](descargar\_datos\_eikon.py):

      Flujo de llamada de funciones: `generate\_diary\_tables()` ➝ `add\_row()` ∈ [historical\_deviation.py](historical\_deviation.py)

      El flujo en [historical\_deviation.py](historical\_deviation.py) donde la llamada se produce desde el archivo [descargar\_datos\_eikon.py](descargar\_datos\_eikon.py) sería el siguiente:

      1. La función que se llama es `add\_row()` que a su vez llama a las siguientes funciones:

         \* Función `\_sanity\_check()`: esta función está diseñada para validar datos nuevos antes de añadirlos a un conjunto de datos históricos existente. Realiza varias comprobaciones para detectar errores comunes que podrían causar problemas o inconsistencias.

         \* Función `\_fill\_derivative\_values()`: extrae ciertas columnas de tipos de interés de bonos de los datos nuevos. Calcula la diferencia de cada una respecto a la tasa de referencia a 10 años, agregando esos valores como nuevas columnas. Luego ajusta el valor del franco suizo multiplicándolo por la tasa del euro.

         \* Función `\_compute\_offset\_dates()` que se utiliza para definir puntos de referencia en el tiempo con el fin de comparar valores de una serie temporal (por ejemplo, para ver cómo se compara un valor actual con el día hábil anterior (ayer), el último viernes, una fecha fija relacionada con una crisis, etc.)

         \* Función `\_compute\_new\_differences()`: opera sobre un DataFrame multi-índice que representa una serie de datos sobre los cuales se calculan diferencias entre nuevos datos introducidos ("new\_chunk\_df") y datos históricos ("historic\_df"). La función modifica "new\_chunk\_df" insertándole las diferencias calculadas directamente en su estructura multi-índice.

         Luego se llaman a varias funciones que permiten calcular medias rodantes y medias expansivas una vez introducidos los nuevos valores. Estas funciones es necesario utilizarlas porque los dataframes no permiten calcular estos valores directamente:

         \* Función `\_update\_rolling\_mean()`

         \* Función `\_update\_rolling\_std()`

         \* Función `\_update\_expanding\_mean()`

         \* Función `\_update\_expanding\_std()`

**## Funcionamiento de la creación de tablas en el documento Word: archivo [generar\_tablas\_word.py](**generar\_tablas\_word.py**)**

La función `fill()` dentro del archivo [generar\_diario.py](generar\_diario.py) para poder completar las tablas en el documento Word, antes tiene que llamar a la función `generate()` dentro del archivo [generar\_tablas\_word.py](generar\_tablas\_word.py) que se ocupa de la generación de estas tablas.

La función `generate()` crea y formatea una tabla compleja en un documento Word a partir de datos almacenados en archivos Feather, usando la estructura y reglas definidas en "table\_dict". Organiza cabeceras, índices y contenido, aplicando estilos y guardando el resultado. Para ello llama a su vez a las siguientes funciones:

1. Función `\_generate\_header()`: construye la fila superior del encabezado de la tabla en Word, donde cada bloque de columnas está representado por una celda fusionada que ocupa el espacio necesario. El texto del bloque se centra y se le aplica un tamaño de fuente adecuado para resaltar.

2. Función `\_fill\_column\_name()`: coloca los nombres de las columnas del DataFrame en la fila correspondiente de la tabla del documento Word, ajustando el formato y la alineación para que la tabla sea clara y estética, considerando si la tabla tiene orientación horizontal o no.

3. Función `\_get\_block\_titles()`: es una función simple que extrae y devuelve los títulos (nombres) de los bloques para usarlos en otras partes del código, como en cabeceras o agrupaciones.

4. Función `\_get\_block\_locs()`: prepara las coordenadas para seleccionar bloques de columnas o filas en un DataFrame con MultiIndex, adaptándose a si la tabla es horizontal o vertical

5. Función `\_fill\_index\_names()`: escribe los nombres de índice en la primera columna de una tabla Word, adaptándose a la orientación horizontal o vertical, y añade separadores visuales si están habilitados para mejorar la legibilidad y organización.

6. Función `\_fill\_content()` se encarga de rellenar el contenido numérico de una tabla dentro de un documento Word usando datos de un DataFrame principal ("table") y un DataFrame de desviaciones ("deviation\_table"). Además, aplica formatos condicionales y estilos, y puede insertar separadores visuales para organizar mejor la tabla. Para todo ello, llama a su vez a las siguientes funciones:

   6.1. Función `\_round()`: recibe un número ("raw\_value") y un número de decimales ("decimals"), y devuelve una cadena ("str\_value") con el número redondeado y formateado para que tenga un aspecto legible y adaptado a formatos numéricos europeos (por ejemplo, usando coma como separador decimal y punto como separador de miles).

   6.2. Función `\_is\_zero()`: chequea si el valor numérico en formato texto es cero (ignorando separadores de miles o decimales representados con comas). Para eso: quita las comas, extrae los caracteres únicos y comprueba que solo quede el carácter "0".

   6.3. Función `\_unit()`: asegura que cada valor numérico tenga la unidad adecuada, ya sea la que se pasa directamente o, si no se especifica, la que corresponde a la fila en la tabla.

   6.4. Función `\_deviation()`: evalúa si un valor se desvía significativamente (más que un umbral) según una tabla de desviaciones. Si es así, pinta la celda con un color que depende de cuánto se desvíe y si el valor es “positivo” o “negativo” según el criterio dado. Para ello llama a su vez a las siguientes funciones:

      \* Función `\_compute\_colors()`: genera un código de color en formato hexadecimal basado en un valor de desviación ("dev\_ratio"), indicando visualmente si la desviación es "buena" o "mala" (según el parámetro "positive").

      \* Función `\_color\_cell()`: colorea una celda en una tabla de Word con un color específico y cambia el color del texto para mantener el contraste y la legibilidad. Para ello llama a la siguiente función:

         \* Función `\_is\_bright()`: evalúa si un color es claro o brillante para poder decidir, por ejemplo, qué color de texto poner encima (negro si es brillante, blanco si es oscuro), ayudando a mejorar la legibilidad

   6.5. Función `\_color\_number()`: cambia el color del texto dentro de una celda específica de una tabla en un documento de Word, según el valor numérico que recibe y un criterio de si valores positivos se consideran buenos o no.

   6.6. Función `\_separate\_cell()`: añade una línea doble en el borde inferior de una celda específica dentro de una tabla de un documento Word, para crear una separación visual entre filas.

\*\*\*

---

**\*\****\*Resumen\****\*\***:

\* [generar\_diario.py](generar\_diario.py) se ocupa sobre todo de 2 cosas: (1) llamar al archivo de descarga de datos vía reuters y (2) rellenar las tablas del Word a partir de la tabla de datos ya descargada.

\* [descargar\_datos\_eikon.py](descargar\_datos\_eikon.py) se ocupa de la descarga, manipulación y análisis de los datos en sí. Para ello necesita llamar al archivo [historical\_deviation.py](historical\_deviation.py) que es el que se ocupa de la manipulación y análisis de los datos.

\*\*\*

---

**## Funcionamiento del flujo de archivos al descargar datos**

\* Los archivos `historic\_{fecha}.feather` contienen datos de cierre de mercado desde el 31/12/2007 hasta el día laborable inmediatamente anterior a `fecha`

\* El archivo `flash.feather` generado hoy contiene los datos de cierre de mercado hasta ayer (es decir, los incluidos en `historic\_{hoy}.feather`) más el dato de mercado del momento en que se ejecuta [generar\_diario.py](generar\_diario.py)

El programa, partiendo del archivo de datos consolidados hasta anteayer (i.e., datos de cierre de mercado hasta t-2 incluidos) que sería `historic\_{ayer}.feather`, crea dos nuevos archivos:

   1. el `historic\_{hoy}.feather` con datos consolidades hasta ayer incluido y

   2. el `flash.feather` con datos consolidades hasta ayer incluido más el dato de mercado del momento en que ejecuta [generar\_diario.py](generar\_diario.py) (sobreescribiendo al archivo flash anterior) que es el que se utiliza cada día para crear el diario en Word

![Flujo](Archivos\_diario.png)

\*\*\*

---

**## \*\*Tutorial\*\*: Cómo añadir una nueva serie al histórico mediante el archivo [reform.py](**reform.py**)**

El archivo a utilizar será [reform.py](reform.py). Concretamente:

1. Dentro del condicional `if \_\_name\_\_ == '\_\_main\_\_':` modificar la lista `new\_series` para que contenga la(s) serie(s) a descargar (tickers de Reuters).

2. Ejecutar el script [reform.py](reform.py).

3. Disfrutar de las nuevas series descargadas :)

**### Flujo de ejecución del archivo [reform.py](**reform.py**)**

Se ejecuta la función `reform()`, que toma como parámetro la lista `new\_series` con los tickers de Reuters a descargar. Esta función a su vez:

   1. Llama a la función `download\_full\_timeseries()`, que asimismo:

      1. Llama a la función `download\_batched\_timeseries()`, que se descarga los datos de reuters "a trozos" y los va guardando en archivos `.csv` temporales.

      2.  Llama a la funcion `unify\_batched\_timeseries()`, que unifica los `.csv` parciales descargados en un único archivo `trim.feather`.

      3. Realiza los cálculos (desviaciones típicas rodantes, medias rodantes, etc.) sobre cada serie llamando a la función `process\_historic()`, creando primero el archivo `trim\_no\_na.feather` sin los N/A para luego crear el archivo consolidado `full.feather` con las nuevas series y los cálculos realizados sobre ellas en distintas columnas.

   2. Se crea una carpeta nueva `historic\_backup` que copia todos los archivos que se encontraran en la carpeta `historic` como copia de seguridad. El programa pide al usuario si desea crear una nueva carpeta `historic\_backup\_2` (en caso de que ya existiera `historic\_backup`) o sobreescribir la que ya existe (`historic\_backup`) con la copia de archivos de la carpera `historic`

   3. Concatena las nuevas series del archivo `full.feather` con las ya existentes en el archivo `flash.feather`, creando el archivo `new\_flash.feather`.

   4. A partir de este nuevo archivo `new\_flash.feather` se generan los archivos `historic\_YYYY-MM-DD.feather` que son los necesarios para generar nuevos diarios.

   (\*) Los archivos `historic\_YYYY-MM-DD.feather` contienen únicamente datos de cierre de mercado. Por ejemplo, el archivo `historic\_2025-06-20.feather` contiene todos los datos de cierre de mercado hasta el 19-Jun-2025 incluido.

   Por otro lado, si hoy es 20-Jun2025, el archivo `flash.feather` - que es el utilizado para crear el diario cada día - contiene los datos de cierre del archivo `historic\_2025-06-20.feather` (datos de cierre hasta el 19-Jun-2025 incluido), más el dato de mercado de la tarde del 20-Jun-2025 del momento en el que se genera el diario.

**### Flujo de ejecución de las funciones auxiliares**

- Función `download\_batched\_timeseries()`:

  1. Divide el rango de fechas a descargar en intervalos más manejables

  2. Descarga los datos de cada intervalo y los va guardando en archivos `.csv` con ayuda de la función `download\_new\_data()`.

- Función `download\_new\_data()`:

  1. Descarga los datos de cierre de la serie y el intervalo de fechas especificado.

  2. Limpia y valida los datos descargados (elimina datos en fin de semana, ordena los datos por fecha, etc.)

- Función `unify\_batched\_timeseries()`: Los datos guardados en los archvios .csv son data frames que se deben unir uno debajo del otro (si corresponden a la misma serie) y uno al lado del otro (si corresponden a la misma serie). Esta función se encarga de esta tarea.

- Función `process\_historic()`: llama a la función `process\_raw\_data()` que es una función propia del módulo [deviation.historical\_deviation](deviation/historical\_deviation.py). Procesa los datos con los cálculos que se utilizarán más adelante para crear las tablas del diario. (**\*\****\*ver sección correspondiente más arriba\****\*\***)

**## \*\*Tutorial\*\*: Cómo añadir una nueva serie al diario word generado (tablas y gráficos)**

1. Archivo [rics.py](rics.py): Dentro del diccionario `diary\_rics` se hay una serie de diccionarios, cada uno de los cuales recoge las series a incluir en un *\*bloque de tabla\**. La clave es el código de la serie y el valor, el texto a mostrar en su lugar en la tabla.

   - Añadir la nueva serie al bloque existente que corresponda.

   - Crear un nuevo bloque que contenga la nueva serie.

```python

"BloqueExistente": {

   "LCOc1": "Petróleo Brent",

   "TRNLTTFMc1": "Gas natural",

   ".BCOM": "Bloomberg commodity index" # nueva serie añadida

},

"NuevoBloque": {

   ".BCOM": "Índice BCOM" # nueva serie añadida con otro nombre a mostrar

},

```

   Este archivo [rics.py](rics.py) es llamado dentro del archivo [descargar\_datos\_eikon.py](descargar\_datos\_eikon.py), que es donde se descargan los datos para el día concreto en el que se genera el diario.

1. Archivo [tables.py](tables.py): Contiene una lista de diccionarios que representan una especificación de las tablas del documento.

   - Si se quiere añadir un nuevo bloque a alguna tabla se debe añadir un nuevo elemento a la lista dentro de la clave `blocks` de la tabla correspondiente.

   - En caso de necesitar una especificación de columnas diferente a la del resto de la tabla, quizá sea necesario modificar el archivo [units.py](units.py)

```python

{

   "title": "Titulo de sección", # puede dejarse a ""

   "heading": "Titulo de tabla",

   "file\_name": "nombre del archivo a partir del cual se construirá",

   "paragraph": None, # función para generar párrafo automático

   "blocks": [

      {"block\_name": "bloque1", "units": units.divisas},

      {"block\_name": "NuevoBloque", "units": units.divisas},

   ],

   "block\_sep": True, # separación visible entre bloques

   "axis": 0, # concatenación de bloques en vertical

}

```

2. Archivo [units.py](units.py): Está compuesto de una serie de variables, cada una de las cuales representa una especificación de las columnas de un bloque de tabla. Varios bloques de tabla (incluso de diferentes tablas) pueden utilizar una misma especificación de columnas.

   - De ser necesario para nuestros propósitos con la nueva serie, puede crearse una nueva.

     - Un ejemplo representativo de esto es el bloque de `Volatilidad europea`. Mostramos a continuación un extracto de cada uno de los ficheros involucrados.

```python

# rics.py

"Volatilidad europea": {

   ".V2TX": "VSTOXX",

},

```

```python

# tables.py

{

   "title": "Renta variable",

   "heading": "Variación diaria de los principales índices europeos",

   "file\_name": "Bolsas\_europeas",

   "paragraph": None,

   "blocks": [

      # ...

      # resto de bloques...

      {"block\_name": "Volatilidad europea", "units": units.volatilidad}, # referencia a units.py

   ],

   "block\_sep": True,

   "axis": 0,

},

```

```python

# units.py

volatilidad = [

   {

      "col\_name": "Último dato",

      "loc": ("no", "absolute", "value"), # índice donde buscar en el flash

      "scale": 1, # valor por el que multiplicar

      "unit": "%", # unidad a mostrar

      "round": 1, # decimales a mantener

      "colors": False, # ¿color neutral o rojo/verde?

      "deviations": False, # sombrear si es anómalo

      "positive\_good": None, # ¿pintamos los positivos de verde o de rojo?

   },

# ...

# resto de columnas

]

```

3. Archivo [paragraphs.py](paragraphs.py): se define el texto que aparecerá en el archivo word. No es necesario modificar nada si no se quiere añadir o quitar nada del texto que saldrá plasmado encima de cada tabla.

4. Archivo [generar\_diario.py](generar\_diario.py): al ejecutarlo, como se muestra al inicio de esta página, se lleva a cabo el siguiente proceso:

   1. proceso de descarga de datos de reuters **\*\*para el día concreto\*\***.

      - Por eso, para nuevas series, es necesario seguir el proceso anterior de añadir nuevas series, para tener los datos pasados **\*\*desde el 31/12/2007\*\***, y no solo el de hoy.

   2. Llamada a la función `fill()`: a través de un bucle `for` que recorre la lista de tablas del archivo [tables.py](tables.py) y *\*renderiza\** cada una de ellas haciendo uso del módulo [generar\_tablas\_word](generar\_tablas\_word.py) (**\*\****\*ver sección correspondiente más arriba\****\*\***)

   3. Llamada a funciones `...\_plots()` definidas dentro de [generar\_diario.py](generar\_diario.py), estas funciones crean los gráficos debajo de las tablas

      - Pueden ser simples, como `divisas\_plots()`, o dobles, como `bolsas\_plots()` o `divisas\_dobles\_plots()`