Ejecución

Requiere PIP instalado.

```
$ pip install -r requirements.txt
$ python main.py
```

Resultados

- El mes que más se ha gastado ha sido Enero (12064.0)
- El mes que menos se ha gastado ha sido Abril (-18933)
- La media de gasto mensual ha sido -1319.1666666666667
- El total de gastos ha sido -279170.0
- El total de ingresos ha sido 251276.0

Desarrollo

He aprovechado el ejercicio para profundizar en el manejo de Pandas, ya que tenía conocimiento muy básico.

He empezado la programación con un "happy path". Dado que las 10 primeras filas del documento contienen datos correctos, al generar el DataFrame he limitado su contenido con `.head(n=10)` para tener un set de datos sin errores.

He profundizado en el control de las excepciones y el porqué de generar una clase propia Error que herede de Exception y que de esta clase hereden todas nuestras clases de excepciones. En la clase Error podemos poner el código común que se ejecute en todas las excepciones, puede ser:

- Formatear el mensaje de texto que se muestra
- Una función para poder detallar el nombre de la clase que lanza la excepción, de forma que nos sea más fácil acotar el error

Otras funciones que pueden ser de interés en esta clase:

- Enviar un email cuando se lance una excepción
- Añadir un registro de log con los datos de la excepción

He ido programando las funcionalidades que se solicitan y forzando algunos errores para ver cómo devuelvo los datos de los errores. También he creado archivos CSV para comprobar el funcionamiento cuando nos falta una columna o cuando un dato viene en blanco.

Estructura

Estoy profundizando en estructurar los ejercicios como si de proyectos complejos se tratara. Evidentemente hacer este ejercicio con paquetes y módulos es exagerado, pero quiero ir profundizando en cómo estructurar proyectos más complejos, de ahí la estructura que presento.

He incluido el uso del paquete os de Python para poder ejecutar el script tanto desde main como desde el directorio del paquete, de forma que pueda hacer pruebas de un único módulo.

Testing

He creado mocks con juegos de prueba en `/test/datasets.py` para poder probar toda la funcionalidad.

Pytest

Los test para PyTest se encuentran en `/test/test_FinancialData_pytest.py`.

Ejecución

```
$ pytest test/test_FinancialData_pytest.py
```

test_checkTwelveColumnsInDataFrame_True

Con este test probamos que el dataset por defecto (el archivo CSV) tenga 12 columnas

test_checkTwelveColumnsInDataFrame_False

En este test mockeamos un dataset con 11 columnas y comprobamos si podemos usarlo

test_checkValues_space_is_converted

En este test mockeamos un dataset con un dato en formato texto y comprobamos que se convierte a cero

Documentación

Herramienta

Después de haber revisado algunas herramientas (Epydoc, Doxygen y MkDocs), he optado por el software Pycco.

Me ha gustado que utiliza MarkDown para formatear, pero no lo hace muy bien. El formato de salida es muy original, con dos columanas. Básicamente lo que hace es sacar los comentarios para ponerlos en la columna de la izquierda a la altura del código al que hacen referencia, que está en la columna de la derecha.

Instalación:

```
$ pip install pycco
```

Uso

Para generar la documentación de un archivo específico:

```
# pycco [ruta]/[archivo].py
$ pycco myPackage/FinancialData.py
```

Para generar la documentación de los archivos contenidos en una carpeta:

```
# pycco [ruta]
$ pycco myPackage
```

Documentos generados

Pycco genera una carpeta `/docs` en la raíz del proyecto con archivos `.html`, uno por cada módulo (archivo) Python.