Report Formatter

Purpose

Ad Ops regularly have to create campaign reports. These are summaries of all line items and creatives. The data for these reports can be pulled down from DFP but in a raw tabular form.

This app aims to automate the beautifying/validating/aggregating of this data.

How to run

This app requires Python to run. If Python is not installed then install it via https://www.python.org/downloads/ for your specific platform.

The app requires the following steps to be completed before running:

1. Run `pip install --upgrade pip`
2. Run `pip install -r requirements.txt` to install all necessary dependencies.
3. Run `python reporter.py`

Note: If running on Windows, you can double click on the *Report Formatter.bat* file to run the steps above for you. If, upon double clicking the above file, a command windows doesn't open up then you may need to open up your own command window, navigate to the root folder of this project and then run `Report Formatter.bat`.

To package up this directory as a zip file, run the `package` script.

Important notes

When running the script, please CAREFULLY read the output to the screen. It will inform you when columns have been removed, when lines have been ignored or altered and it will give you a checksum at the end to make sure that the totals have been correctly calculated.

This tool has been built to expediate the formatting of reports. Whilst it has been written with care and caution, it is not guaranteed to be correct in all circumstances and so FULLY depends on you, the user, to ensure that the resultant report looks right, does not contain errors, and has not removed or altered data incorrectly.

How it works

When the app runs, it will look for Excel files ('.xlsx') in the **inputs** folder. It will attempt to transform each file into a report and it will then save the result in the **outputs** folder.

When transforming a report, a number of errors can occur such as missing column names or invalid values. Upon such an error, the app will write out a message to the screen and will then skip the current input file being worked on. Whenever such an event occurs, this should be communicated back to the user via a message on the screen.

Code Design

This section is for those that are interested in how this app works or who wish to alter/amend/fix it.

Structure

The python source files (found in **src**) have the following purposes:

- common.py: shared methods across all modules, typically logging functions.
- exhelp.py: useful Excel helper methods not offered by the openpyxl module.
- cleaners.py: methods that take in a pandas dataframe, apply some cleaning function and then return the resultant dataframe
- validators.py: methods that take in a pandas dataframe, apply some function (DataFrame => Boolean) and return the result.
- reporter.py: this is the main file that utilises the above to take input reports and produce formatted output reports.

Design

The purpose of this tool is quite specific; the task is not to build a fully abstracted report formatting tool. That said, some abstractions were useful for developing new features and altering existing features.

The flow of reporter.py is:

1. find an XLSX file in the input folder and read the data into a dataframe;
2. check that it is "valid" according to validators specified in `validators.py`;
3. clean up the data using cleaners specified in `cleaners.py`;
4. write the dataframe data to an output Excel file with various markup tags;
5. use the markup tags to style various elements of the sheet
6. replace remaining tags with tag-specific values i.e. subtotals and totals
7. save the workbook

Only `reporter.py` should know about specific column/data specifics. All other methods have been written generically such that they can be used even if the data changes.

The biggest risk in using this tool is that data could be lost or altered without the end user realising. As a result there is a lot of logging and checking the user is paying attention via prompts.