**README**

Please carefully read the following documentation, which clarifies the sequencing and logic behind the codebase. Each step builds on the last, such that incompletely or not following a step will prevent following steps from being successfully executed.

**Step 1: Inputs and folder structure**

The basic input for the scraper is a data file “00 urls\_in.xlsx”. The sheet ‘urls’ contains one row per firm (or website). In Compustat, firms are represented by unique GVKEYs, but this indicator can be anything as long as it offers a unique identifier for each unit of analysis.

In this example, each row contains the GVKEY (column ‘gvkey’), the name of the firm (column ‘name’; this information is not used in the scripts and as such is not required), the GVKEY with a slash added in front (‘gvkey\_withslash’; since some programmes will omit the leading zeroes. Our code contains a few lines that will always ensure adding the required leading zeroes for GVKEYs—this can be altered or removed in case leading zeroes are not required), the first year in which the firm occurred in the data (‘firstyear’), and the last year that the firm occurred in the data (‘lastyear’).

In this example, we'll only try to collect an archived snapshot for a single website for the years 2015 and 2016. Hence, the ‘firstyear’ and ‘lastyear’ columns are set to 2015 and 2016.

Please note that there need to be four, initially empty, folders in the same directory as the script: ‘HTML’, ‘TXT\_uncleaned’, ‘TXT\_cleaned’ and ‘TXT\_combined’. The first will host the downloaded HTML pages. The second will contain the text files from each individual page before any further cleaning or selection. The third will contain the cleaned and selected texts, and the fourth the cleaned and selected texts aggregated to the GVKEY/year.

The code was written for Python version 3 and was tested on Windows in Python 3.9.0 (but should also work on Apple products and more recent versions of Python). The code requires Python software and uses a number of packages that will first need to be installed (see <https://packaging.python.org/en/latest/tutorials/installing-packages/>).

Specifically, the following packages are used and may need to be installed. The versions of these packages that were used at the time of writing are shown between parentheses. If you want to install these specific package versions (rather than the latest package version) when installing, please use the command “pip install package\_name==version\_number”. For example, “pip install bs4==0.0.1” installs bs4 version 0.0.1.

* bs4 (version 0.0.1)
* datetime (version 5.2)
* langdetect (version 1.0.9)
* numpy (version 1.24.3)
* openai (version 0.27.8)
* pandas (version 2.0.2)
* requests (version 2.31.0)
* tiktoken (version 0.4.0)
* url\_normalize (version 1.4.3)

**Step 2: Collecting frontpages**

The script ‘01 Collect frontpages.py’ is the first script that you should run after completing the core set-up.

This script will work from the ‘urls\_input.xlsx’ file and run through the years 1996 (the earliest possible date) until 2020 (the endpoint of our current Compustat dataset) for the specified URLs. In this case, due to the limited range for the ‘firstyear’ and ‘lastyear’ columns, only 2015 and 2016 will be scraped for an archived page.

The script loops through each year in the year list, visiting only the websites of GVKEYs that are in Compustat in that given year. The scraper will by default collect the page nearest to a specified reference point, and as such setting the date to e.g. January 1st might collect pages in December of the prior year—which then get dropped as they're not in the focal year. We therefore take as the reference point the middle day in the year. This will ensure that if there is any page in the year that it will be grabbed.

The script will create two new CSV files: ‘scrapedURLs.csv’ which will serve as the central progress tracker, and ‘URLs\_1\_deeper.csv’ which will contain the list of URLs that are found on the frontpages and that point to the same domain as the frontpage.

The script will download the HTML pages if available to the ‘HTML’ folder that was created in the previous step. Although we already cleaned the URLs upfront, the script also incorporates some basic cleaning in the script here to prevent issues with future use. In the rare case that the URL is entirely invalid and cannot be normalized, the script will not continue for that GVKEY/year and will write the respective observation to the tracking file. If there are no archived pages, it will stop scraping (there is nothing to scrape) not continue and will write this to the tracking file. If there is an archived page, it will continue as long as the closest snapshot to the reference date is within the focal year of data collection. If the archived page falls outside the year, it will not continue and will write this to the tracking file.

Any pages that were not collected can be seen in the ‘scrapedURLs.csv’ file; they will not have the ‘Collected’ status for the ‘valid\_scrape’ column.

The script also checks all URLs on these scraped frontpages, processing and cleaning them in order to keep only those that point to the same domain as the frontpage and which do not point to e.g. pictures or programmes. All these cleaned URLs then get added to the list of URLs to collect in the second script.

**Step 3: Collecting further pages**

If you’re only interested in frontpages then you can continue to step 3. If you want to collect the pages that are one click deeper on the front page, however, the script ‘02 Collect further pages.py’ is the second script that you should run.

This script will work from the ‘URLs\_1\_deeper.csv’ file and will visit each of these URLs to access their archived version. As it runs, the script will expand the ‘scrapedURLs.csv’ file. As before, collected pages are downloaded to the ‘HTML’ folder while uncollected pages will be tracked in the ‘scrapedURLs.csv’ file. The sequence of steps and cleaning is identical to the above, except that further URLs on the respective pages do not get processed. See the FAQ and the script ‘99 Scrape further levels down.py’ for information how to move further levels down if so desired.

It is worth noting two issues that emerged in this example (see ‘scrapedURLs.csv’). First, the page with identifier 2016\_023804\_1\_5 did not have any snapshot available and thus was not collected. This is normal. Second, there was no archived website in the year 2016 (with identifier 2016\_023804\_0\_0).

**Step 4: Archive the HTML pages**

At this stage, you should convert the ‘HTML’ folder into a .zip archive. Although it is possible to simply loop through files in the ‘HTML’ folder, we have found it to be more efficient and storage-friendly to archive the individual files. For example, Windows tends to slow down as a whole when there are folders containing many individual files. After archiving the ‘HTML’ folder into a .zip archive called ‘HTML.zip’, you can continue with the following steps. It is safe to remove the ‘HTML’ folder after this.

**Step 5: Process and select the HTML pages**

The third script, ‘03 Convert HTML to plaintext.py’ will work from the ‘HTML.zip’ archive. It will go through each HTML file in this .zip file and parse textual data from it. It will write these resulting text files to the folder ‘TXT\_uncleaned’ that was created in the set-up.

This script combines HTML processing with the creation of a file containing titles and page URLs to go into classification in the next step. In terms of HTML processing, it cleans up a variety of scripts and other content (hidden content, headers, footers, et cetera). This information gets stored in the‘input\_categorization\_allpages.csv’ file.

**Step 6 (optional): Classify pages using GPT**

In this optional step, the script ‘04 GPT application.py’ will use the information written to the ‘input\_categorization\_allpages.csv’ file (containing in particular the title of the page and the path in the URL for those pages that met the selection criteria) as input into the API of OpenAI's GPT. Here, for cost and efficiency reasons, we used GPT3.5 although any model can be used depending on your preferences.

Also note that this step is entirely optional; by default and unless your personal API key gets pasted into the code below, a placeholder file will be created which will not lead to any further selection. Specifically, the files ‘categorization\_applied.csv’ and ‘categorization\_errors.csv’ will be created, with the former containing the classifications that the GPT model will identify for the pages and the latter containing information on any errors that may emerge during the process.

In this example, using our own API key, yields the 'categorization\_applied.csv' file which contains the relevant page category codes. The errors file is empty, as there were no errors. When you do not want to use this categorization step, you can simply continue to the next step.

**Step 7: Archive the TXT\_uncleaned folder**

Before running the following script, it is necessary to archive the ‘TXT\_uncleaned folder into a .zip archive called 'TXT\_uncleaned ', for the same reasons as discussed above.

**Step 8: Clean and select texts**

The final script, ‘05 Clean and select.py’ applies several pre-defined selection and cleaning steps. Specifically, it filters out duplicate pages, pages that were *a priori* identified as invalid, pages that were classified by GPT as not relevant types, and non-English pages and cleans up the remaining texts by removing (sub-)sentences contained in the “exclusion\_list.xlsx” file as well as removing any texts that are ten words or shorter after this cleaning. These texts get written to the ‘TXT\_cleaned’ folder. It also combines these pages into a single observation per GVKEY/year into the ‘TXT\_combined’ folder. This all gets tracked into the ‘metadata.csv’ file, and if one wishes to make different selection or cleaning decisions it is possible to alter the settings found at the top of the Python script.

In our case, the pages ‘2015\_023804\_1\_2’ and ‘2015\_023804\_1\_8’ get excluded due to them being classified as legal pages (‘drop\_pagetype’ == 1 in ‘metadata.csv’). Additionally, ‘2015\_023804\_1\_9’ gets removed as it only contains nine words after cleaning.

In sum, the set-up is as follows:

1. Create empty folders called 'HTML', 'TXT\_uncleaned, ‘TXT\_cleaned’ and 'TXT\_combined'.
2. Run the ‘01 Collect frontpages.py’ script to collect the frontpages.
3. Run the ‘02 Collect further pages.py’ script to collect all pages one click deeper on the domain.
4. Archive the files in the 'HTML' folder to a .zip file called 'HTML.zip'.
5. Run the ‘03 Convert HTML to plaintext.py’ script to process the files in ‘HTML.zip’.
6. **Optional:** Run the ‘04 GPT application.py’ script to classify the selected pages for further selection.
7. Archive the files in the 'TXT\_uncleaned folder to a .zip file called 'TXT\_uncleaned.zip'.
8. Run the ‘05 Clean and select.py’ script to apply selection and cleaning of the texts into the ‘TXT\_cleaned’ folder and combine the selected pages to one plaintext file per GVKEY/year into the ‘TXT\_combined’ folder.