sec_scrapper Documentation

Alexandre Bondoux

Oct 18, 2019

Contents:

This library provides a high level access to the SEC data. The main goal is to create virtual portfolios based on parsed 10-X filings.

Processing the reports is time consuming and the user should be ready to scale up/out the work in order to make iterations manageable.

View the GitHub repo for more information and sample code.

CHAPTER 1

History of the project

The secScraper project was done as part of my Data Engineering fellowship at Insight in Fall 2019. Over 3 weeks, that project was started from scratch to where it stands.

CHAPTER 2

secScraper package

Here is the documentation of all the modules in the secScraper package.

2.1 secScraper.display module

secScraper.display.**diff_vs_benchmark**(*pf_values*, *index_name*, *index_data*, *diff_method*, *s*, *norm_by_index=False*)

Plot a portfolio vs an index.

Parameters

- **pf_values** – Value of the portfolio over time.
- **index_name** – Name of the index.
- **index_data** – Daily value of the index.
- **s** – Settings dictionary.

Returns void

secScraper.display.**diff_vs_benchmark_ns**(*pf_values*, *index_name*, *index_data*, *diff_method*, *s*, *norm_by_index=False*)

Plot a portfolio vs an index.

Parameters

- **pf_values** – Value of the portfolio over time.
- **index_name** – Name of the index.
- **index_data** – Daily value of the index.
- **s** – Settings dictionary.

Returns void

secScraper.display.**diff_vs_stock**(*qtr_metric_result*, *ticker_data*, *ticker*, *s*, *method='diff'*)

Display the calculated data for a given ticker across the time_range that was specified.

Parameters

- **qtr_metric_result** – Dictionary containing the data to plot
- **ticker_data** – Daily stock value for the ticker considered
- **ticker** – Company ticker on the US stock exchange
- **s** – Settings dictionary
- **method** – Specify if a difference between two reports or an analysis of each report.

Returns void

secScraper.display.**histogram_width**(*qtr_metric_result*, *metrics*, *s*)

secScraper.display.**plot_diff_vs_benchmark**(*benchmark*, *bin_data*, *index_name*, *s*)

secScraper.display.**plot_diff_vs_stock**(*benchmark*, *metric_data*, *ticker*, *s*, *method='diff'*)

secScraper.display.**run_from_ipython**()
Check if the script is run from command line or from a Jupyter Notebook.

Returns bool that is True if run from Jupyter Notebook

secScraper.display.**update_ax_diff_vs_benchmark**(*ax*, *benchmark*, *bin_data*, *index_name*, *s*, *ylim*, *m*)

2.2 secScraper.metrics module

secScraper.metrics.**composite_index**(*data*)
Create a composite index based on the sentiment analysis based on Loughran and McDonald's dictionary and script.

Parameters **data** – String to analyse.

Returns List of values. See unused variable OUTPUT_FIELDS in the source, there is a lot.

secScraper.metrics.**cosine_similarity**(*vector1*, *vector2*)

secScraper.metrics.**diff_cosine_tf**(*str1*, *str2*)
Calculates the Cosine TF similarity between two strings.

Parameters

- **str1** – First string.
- **str2** – Second string.

Returns float in the [0, 1] interval

secScraper.metrics.**diff_cosine_tf_idf**(*str1*, *str2*)
Calculates the Cosine TF-IDF similarity between two strings.

Parameters

- **str1** – First string.
- **str2** – Second string.

Returns float in the [0, 1] interval

secScraper.metrics.**diff_edit_distance**(*str1*, *str2*)

secScraper.metrics.**diff_gfg_editDistDP**(*str1*, *str2*)

secScraper.metrics.**diff_jaccard**(*str1*, *str2*)
Calculates the Jaccard similarity between two strings.

Parameters

- **str1** – First string.
- **str2** – Second string.

Returns float in the [0, 1] interval

secScraper.metrics.**diff_minEdit**(*str1*, *str2*)
Calculates the minEdit similarity between two strings. This is word based. WARNING: VERY SLOW BEYOND ~10,000 CHAR TO COMPARE.

Parameters

- **str1** – First string.
- **str2** – Second string.

Returns float in the [0, 1] interval

secScraper.metrics.**diff_simple**(*str1*, *str2*)
Calculates the simple difference similarity between two strings. This is character based. WARNING: VERY SLOW BEYOND ~10,000 CHAR TO COMPARE.

Parameters

- **str1** – First string.
- **str2** – Second string.

Returns float in the [0, 1] interval

secScraper.metrics.**diff_sk_cosine_tf**(*str1*, *str2*, *stop_words*)

secScraper.metrics.**diff_sk_cosine_tf_idf**(*str1*, *str2*, *stop_words*)

secScraper.metrics.**sing_sentiment**(*text*, *lm_dictionary*)
Run the Loughran and McDonald's sentiment analysis on a string.

Parameters

- **text** – String to analyze.
- **lm_dictionary** – Sentiment dictionary

Returns Quite a few fields.

2.3 secScraper.parser module

secScraper.parser.**clean_first_markers**(*res*)
In the event that a ToC was found, this will remove every first entry in the values of res. That means that all the location related to the titles in the ToC will be removed.

Parameters **res** – dict, keys are sections to parse and contain the locations where the titles were found in the text.

Returns Filtered version of res without the ToC locations

class secScraper.parser.**stage_2_parser**(*s*)
Bases: object

Parser object. Acts on Stage 1 data.

parse(*parsed_report*, *verbose=False*)
Parse the text in a report. The text of each section will be placed in a different dict key.

Parameters

- **parsed_report** – the text, as a giant str
- **verbose** – Increase the amount of printing to the terminal

Returns dict containing the parsed report with all the text by section. Metadata is in '0'

2.4 secScraper.post_processing module

secScraper.post_processing.**build_portfolio**(*pf_values*, *lookup*, *stock_data*, *s*)

secScraper.post_processing.**buy_all_pf**(*qtr*, *funds*, *pf*, *lookup*, *stock_data*, *method*)
Allocate a given amount of money to a quarterly portfolio. Method can be balanced (weighted by market cap) or unbalanced (each stock gets the same amount of money).

secScraper.post_processing.**calculate_portfolio_value**(*pf_scores*, *pf_values*, *lookup*, *stock_data*, *s*, *balancing='balanced'*, *verbose=False*)
Calculate the value of a portfolio, in equal weight and balanced weight (by market cap) mode. The value is written to pf_scores (in the inputs).

Parameters

- **pf_scores** – dict containing all the scores for all companies
- **pf_values** – dict containing the value of a portfolio
- **lookup** – lookup dict
- **stock_data** – dict of the stock data
- **s** – Settings dictionary

Returns dict pf_scores

secScraper.post_processing.**check_pf_value**(*pf_values*, *s*)

secScraper.post_processing.**create_metric_scores**(*cik_scores*, *lookup*, *stock_data*, *s*)

secScraper.post_processing.**dump_cik_scores**(*cik_scores*, *s*)

secScraper.post_processing.**dump_master_dict**(*master_dict*, *s*)

secScraper.post_processing.**dump_pf_values**(*pf_values*, *s*)

secScraper.post_processing.**get_pf_value**(*pf_scores*, *m*, *mod_bin*, *qtr*, *lookup*, *stock_data*, *s*)
Get the value of a portfolio.

Parameters

- **pf_scores** – dict containing all the scores for all companies
- **m** – metric
- **mod_bin** – bin considered
- **qtr** – qtr
- **lookup** – lookup dict

- **stock_data** – dict of the stock data
- **s** – Settings dictionary

Returns

secScraper.post_processing.**get_share_price**(*cik*, *qtr*, *lookup*, *stock_data*, *verbose=False*)
Get the price of a share.

Parameters

- **cik** – CIK
- **qtr** – qtr
- **lookup** – lookup dict
- **stock_data** – dict of the stock data
- **verbose** – self explanatory

Returns share_price, market_cap, flag_price_found

secScraper.post_processing.**initialize_portfolio**(*metric_scores*, *s*)

secScraper.post_processing.**make_quintiles**(*qtr_data*, *s*, *winsorize=0.01*, *verbose=False*)

secScraper.post_processing.**metrics_correlation**(*metric_scores*, *s*)

secScraper.post_processing.**remove_cik_without_price**(*pf_scores*, *lookup*, *stock_data*, *s*, *verbose=False*)
So far, we have not checked if we had a stock price available for that all CIK. This function removes the CIK for which we have no price. < 10% of them are dropped.

Parameters

- **pf_scores** – dict
- **lookup** – lookup dict
- **stock_data** – dict of the stock data
- **s** – Settings dictionary
- **verbose** –

Returns outputs more stuff

secScraper.post_processing.**sell_all_pf**(*qtr*, *pf*, *lookup*, *stock_data*)
Sell all the stocks in a portfolio. In practice, we just collect the value of the pf with the new updated share prices.

2.5 secScraper.pre_processing module

class secScraper.pre_processing.**ReadOnlyDict**
Bases: dict

Simple dictionary class that makes it read-only. This applies to the settings dictionary most likely.

set_read_state(*read_only=True*)
Allow or deny modifying dictionary.

Parameters **read_only** – bool to set the state of the dictionary

Returns

secScraper.pre_processing.**check_report_continuity**(*quarterly_submissions*, *s*, *verbose=False*)

Verify that the sequence of reports for the various qtr is 0-...0-1-...-1-0-...-0. In other words, once you are listed you only have one and only one report per quarter until you are delisted.

Parameters

- **quarterly_submissions** –
- **s** –

Returns

secScraper.pre_processing.**check_report_type**(*quarterly_submissions*, *qtr*)

Verify that all the reports in quarterly_submissions were published at the right time based on their type. A 10-K is supposed to be published and only published in Q1. A 10-Q is supposed to be published and only published in Q2, Q3 or Q4.

Parameters

- **quarterly_submissions** – dictionary of reports published, by qtr. There should only be one report per qtr
- **qtr** – A given qtr

Returns void but will raise if the report in [0] was not published at the right time.

secScraper.pre_processing.**dump_tickers_crsp**(*path_dump_file*, *tickers*)

Dump all tickers to a file - should not be useful anymore.

Parameters

- **path_dump_file** – path for csv dump
- **tickers** – all the tickers to dump.

Returns void

secScraper.pre_processing.**filter_cik_path**(*file_list*, *s*)

Filter out all the reports that are not of the considered type. The considered type is available in the settings dictionary.

Parameters

- **file_list** –
- **s** –

Returns

secScraper.pre_processing.**find_first_listed_qtr**(*quarterly_submissions*, *s*)

Finds the first qtr for which the company published at least one report.

Parameters

- **quarterly_submissions** – dictionary of submissions indexes by qtr
- **s** – Settings dictionary

Returns bool for success and first qtr when the company was listed.

secScraper.pre_processing.**intersection_lookup_stock**(*lookup*, *stock*)

Finds the intersection of the set of CIKs contained in the lookup dictionary and the CIKs contained in the stock database. This is part of the steps taken to ensure that we have bijections between all the sets of CIKs for all external databases.

Parameters

- **lookup** – lookup dictionary
- **stock** – stock data, organized in a dictionary with tickers as keys.

Returns both dictionaries with only the intersection of CIKs left as keys.

secScraper.pre_processing.**intersection_sec_lookup**(*cik_path*, *lookup*)
Finds the intersection of the set of CIKs contained in the cik_path dictionary and the CIKs contained in the lookup table. This is part of the steps taken to ensure that we have bijections between all the sets of CIKs for all external databases.

Parameters

- **cik_path** – Dictionary of paths organized by CIKs
- **lookup** – lookup table CIK -> ticker

Returns both dictionaries with only the intersection of CIKs left as keys.

secScraper.pre_processing.**is_permanently_delisted**(*quarterly_submissions*, *qtr*, *s*)
Check if a company is permanently delisted starting from a given qtr. This function is not great, I should have made a single function that finds the first qtr for which a company is listed and the qtr for which it became delisted, if ever.

Parameters

- **quarterly_submissions** –
- **qtr** – a given qtr
- **s** – Settings dictionary

Returns bool assessing whether or not it is permanently delisted after the given qtr

secScraper.pre_processing.**load_cik_path**(*s*)
Find all the file paths and organize them by CIK.

Parameters **s** – Settings dictionary

Returns Dictionary of paths with the keys being the CIK.

secScraper.pre_processing.**load_index_data**(*s*)
Loads the csv files containing the daily historical data for the stock market indexes that were selected in s.

Parameters **s** – Settings dictionary

Returns dictionary of the index data.

secScraper.pre_processing.**load_lookup**(*s*)
Load the CIK -> Lookup table.

Parameters **s** – Settings dictionary

Returns Lookup table in the form of a dictionary.

secScraper.pre_processing.**load_stock_data**(*s*, *penny_limit=0*, *verbose=True*)
Load all the stock data and pre-processes it. WARNING: Despite all (single process) efforts, this still takes a while. Using map seems to be the fastest way in python for that O(N) operation but it still takes ~ 60 s on my local machine (1/3rd reduction)

Parameters **s** – Settings dictionary

Returns dict stock_data[ticker][time stamp] = (closing, market cap)

secScraper.pre_processing.**paths_to_cik_dict**(*file_list*, *unique_sec_cik*)
Organizes a list of file paths into a dictionary, the keys being the CIKs. unique_sec_cik is used to initialize the cik_dict.

Parameters

- **file_list** – unorganized list of paths
- **unique_sec_cik** – set of all unique CIK found

Returns a dictionary containing all the paths, organized by CIKs

secScraper.pre_processing.**review_cik_publications**(*cik_path*, *s*)
Filter the CIK based on how many publications there are per quarter This function reviews all the CIK to make sure there is only 1 publication per qtr It provides a few hooks to correct issues but these have not been implemented. Around 10 % of the CIK seem to have problems at one point or another.

Parameters

- **cik_path** –
- **s** – Settings dictionary

Returns A filtered version of the cik_path dictionary - only has the keys that passed the test.

secScraper.pre_processing.**unique_cik**(*path_list*)
Identify all unique CIK in a path list.

Parameters **path_list** – list of path, most likely obtain from a recursive glob.glob

Returns list of unique CIK found

2.6 secScraper.processing module

2.7 secScraper.qtrs module

secScraper.qtrs.**create_list_url_master_zip**(*list_qtr*)
Generates the URLs for the master indexes for a list of qtr.

Parameters **list_qtr** – list of qtr of interest

Returns list of URLs

secScraper.qtrs.**create_qtr_list**(*time_range*)
From a given time_range, create the list of qtr contained in it. Includes both qtr in the time_range. time_range is of the form [(year, QTR), (year, QTR)].

Parameters **time_range** – a list of two tuples representing the start and finish qtr

Returns a list of all the qtr included in the time_range.

secScraper.qtrs.**display_download_stats**(*stats*)
A better way to display the downloading stats and make sure there is enough space on the disk for the download. If not, consider increasing your EBS size on AWS. If you fill up the disk, that sucks. Make sure there are some sacrificial files in your current terminal folder so you can easily regain control. Anyway.

Parameters **stats** – download stats

Returns void

secScraper.qtrs.**doc_url_to_FilingSummary_url**(*end_url*)
Convert a document url to the url of its xml summary. WARNING: Not all files have a filing summary. 10-Q and 10-K do.

Parameters **end_url** – end url as found in the master index

Returns URL of the xml document that has the section info about the file.

secScraper.qtrs.**doc_url_to_filepath**(*submission_date*, *end_url*)
This one is about onverting a URL into a local file path for download.

Parameters

- **submission_date** – date the document was submitted
- **end_url** – end url as found in the master index

Returns local download path for the html file

secScraper.qtrs.**is_downloaded**(*filepath*)
Checks if a file at a given path already exists or not.

Parameters **filepath** – string that represents a local path

Returns bool

secScraper.qtrs.**master_url_to_filepath**(*url*)
Transforms a master URL into a local file path. This needs to be refactored to be driven from a settings dict.

Parameters **url** – Initial EDGAR URL

Returns local path

secScraper.qtrs.**parse_index**(*path*, *doc_types*)
Parses one master index and returns the URL of all the interesting documents in a dictionary.

Parameters

- **path** – string representing the path of the master index
- **doc_types** – types of documents we are interested in, as a list

Returns dict containing the end url for each type of doc we are interested in.

secScraper.qtrs.**previous_qtr**(*qtr*, *s*)
For once, a self-explanatory function name! Calculate what the previous qtr is in s['list_qtr']

Parameters

- **qtr** – given qtr
- **s** – Settings dictionary

Returns previous qtr in s['list_qtr']

secScraper.qtrs.**qtr_to_day**(*qtr*, *position*, *date_format='string'*)
Dumb function that returns the first or last day in a quarter. Two options for the output type: string or datetime.

Parameters

- **qtr** – given qtr
- **position** – specify 'first' or 'last' day of the qtr. By default last is the 31st so it might not exist.
- **date_format** – 'string' or 'datetime'. Specifies the output type.

Returns result. Read the above

secScraper.qtrs.**qtr_to_master_url**(*qtr*)
Build the URL for the master index of a given quarter.

Parameters **qtr** – given qtr

Returns string that represents the URL

secScraper.qtrs.**unzip_file**(*path*)
Create an unzipping function to be run by a pool of workers.

Parameters **path** – string representing the path of a zip file

Returns void

secScraper.qtrs.**yearly_qtr_list**(*time_range*)
Give all the qtr of each year included in a given time_range. Both start and end qtr are included.

Parameters **time_range** – start and finish qtr

Returns list of lists

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

I

L

M

P

Q

R

S

U

Y