advertools Documentation

Release 0.13.1

Elias Dabbas

Jul 22, 2022

CONTENTS

CHAPTER

ONE

ONLINE MARKETING PRODUCTIVITY AND ANALYSIS TOOLS

Crawl websites, Generate keywords for SEM campaigns, create text ads on a large scale, analyze multiple SERPs at once, gain insights from large social media posts, and get productive as an online marketer.

If these are things you are interested in, then this package might make your life a little easier.

Webinar: Creating SEM campaigns on a large scale - Wednesday April 20, 2022

New: `crawl_headers` Function for crawling a known list of URLs with the HEAD method only
New: SEO crawler has new options for following links, include/exclude URL params and/or URL regex.
New: `reverse_dns_lookup` Function for getting host information on a list of IP addresses

1.1 `advertools`: productivity & analysis tools to scale your online marketing

A digital marketer is a data scientist.
Your job is to manage, manipulate, visualize, communicate, understand, and make decisions based on data.

You might be doing basic stuff, like copying and pasting text on spread sheets, you might be running large scale automated platforms with sophisticated algorithms, or somewhere in between. In any case your job is all about working with data.

As a data scientist you don't spend most of your time producing cool visualizations or finding great insights. The majority of your time is spent wrangling with URLs, figuring out how to stitch together two tables, hoping that the dates, won't break, without you knowing, or trying to generate the next 124,538 keywords for an upcoming campaign, by the end of the week!

`advertools` is a Python package that can hopefully make that part of your job a little easier.

1.1.1 Installation

```
pip install advertools
# OR:
pip3 install advertools
```

1.1.2 SEM Campaigns

The most important thing to achieve in SEM is a proper mapping between the three main elements of a search campaign

Keywords (the intention) -> **Ads** (your promise) -> **Landing Pages** (your delivery of the promise) Once you have this done, you can focus on management and analysis. More importantly, once you know that you can set this up in an easy way, you know you can focus on more strategic issues. In practical terms you need two main tables to get started:

- Keywords: You can generate keywords (note I didn't say research) with the *kw_generate* function.
- Ads: There are two approaches that you can use:
  - Bottom-up: You can create text ads for a large number of products by simple replacement of product names, and providing a placeholder in case your text is too long. Check out the ad_create function for more details.
  - Top-down: Sometimes you have a long description text that you want to split into headlines, descriptions and whatever slots you want to split them into. ad_from_string helps you accomplish that.
- Tutorials and additional resources
  - Get started with Data Science for Digital Marketing and SEO/SEM
  - Setting a full SEM campaign for DataCamp's website tutorial
  - Project to practice generating SEM keywords with Python on DataCamp
  - Setting up SEM campaigns on a large scale tutorial on SEMrush
  - Visual tool to generate keywords online based on the *kw_generate* function

1.1.3 SEO

Probably the most comprehensive online marketing area that is both technical (crawling, indexing, rendering, redirects, etc.) and non-technical (content creation, link building, outreach, etc.). Here are some tools that can help with your SEO

- SEO crawler: A generic SEO crawler that can be customized, built with Scrapy, & with several features:
  - Standard SEO elements extracted by default (title, header tags, body text, status code, reponse and request headers, etc.)
  - CSS and XPath selectors: You probably have more specific needs in mind, so you can easily pass any selectors to be extracted in addition to the standard elements being extracted
  - Custom settings: full access to Scrapy's settings, allowing you to better control the crawling behavior (set custom headers, user agent, stop spider after x pages, seconds, megabytes, save crawl logs, run jobs at intervals where you can stop and resume your crawls, which is ideal for large crawls or for continuous monitoring, and many more options)
  - Following links: option to only crawl a set of specified pages or to follow and discover all pages through links
- robots.txt downloader A simple downloader of robots.txt files in a DataFrame format, so you can keep track of changes across crawls if any, and check the rules, sitemaps, etc.

- XML Sitemaps downloader / parser An essential part of any SEO analysis is to check XML sitemaps. This is a simple function with which you can download one or more sitemaps (by providing the URL for a robots.txt file, a sitemap file, or a sitemap index
- SERP importer and parser for Google & YouTube Connect to Google's API and get the search data you want. Multiple search parameters supported, all in one function call, and all results returned in a DataFrame
- Tutorials and additional resources
  - A visual tool built with the `serp_goog` function to get SERP rankings on Google
  - A tutorial on analyzing SERPs on a large scale with Python on SEMrush
  - SERP datasets on Kaggle for practicing on different industries and use cases
  - SERP notebooks on Kaggle some examples on how you might tackle such data
  - Content Analysis with XML Sitemaps and Python
  - XML dataset examples: news sites, Turkish news sites, Bloomberg news

1.1.4 Text & Content Analysis (for SEO & Social Media)

URLs, page titles, tweets, video descriptions, comments, hashtags are some exmaples of the types of text we deal with. `advertools` provides a few options for text analysis

- Word frequency Counting words in a text list is one of the most basic and important tasks in text mining. What is also important is counting those words by taking in consideration their relative weights in the dataset. `word_frequency` does just that.
- URL Analysis We all have to handle many thousands of URLs in reports, crawls, social media extracts, XML sitemaps and so on. `url_to_df` converts your URLs into easily readable DataFrames.
- Emoji Produced with one click, extremely expressive, highly diverse (3k+ emoji), and very popular, it's important to capture what people are trying to communicate with emoji. Extracting emoji, get their names, groups, and sub-groups is possible. The full emoji database is also available for convenience, as well as an `emoji_search` function in case you want some ideas for your next social media or any kind of communication
- extract_ functions The text that we deal with contains many elements and entities that have their own special meaning and usage. There is a group of convenience functions to help in extracting and getting basic statistics about structured entities in text; emoji, hashtags, mentions, currency, numbers, URLs, questions and more. You can also provide a special regex for your own needs.
- Stopwords A list of stopwords in forty different languages to help in text analysis.
- Tutorial on DataCamp for creating the `word_frequency` function and explaining the importance of the difference between absolute and weighted word frequency
- Text Analysis for Online Marketers An introductory article on SEMrush

1.1.5 Social Media

In addition to the text analysis techniques provided, you can also connect to the Twitter and YouTube data APIs. The main benefits of using `advertools` for this:

- Handles pagination and request limits: typically every API has a limited number of results that it returns. You have to handle pagination when you need more than the limit per request, which you typically do. This is handled by default
- DataFrame results: APIs send you back data in a formats that need to be parsed and cleaned so you can more easily start your analysis. This is also handled automatically
- Multiple requests: in YouTube's case you might want to request data for the same query across several countries, languages, channels, etc. You can specify them all in one request and get the product of all the requests in one response
- Tutorials and additional resources
- A visual tool to check what is trending on Twitter for all available locations
- A Twitter data analysis dashboard with many options
- How to use the Twitter data API with Python
- Extracting entities from social media posts tutorial on Kaggle
- Analyzing 131k tweets by European Football clubs tutorial on Kaggle
- An overview of the YouTube data API with Python

1.1.6 Conventions

Function names mostly start with the object you are working on, so you can use autocomplete to discover other options:

`kw_`: for keywords-related functions
`ad_`: for ad-related functions
`url_`: URL tracking and generation
`extract_`: for extracting entities from social media posts (mentions, hashtags, emoji, etc.)
`emoji_`: emoji related functions and objects
`twitter`: a module for querying the Twitter API and getting results in a DataFrame
`youtube`: a module for querying the YouTube Data API and getting results in a DataFrame
`serp_`: get search engine results pages in a DataFrame, currently available: Google and YouTube
`crawl`: a function you will probably use a lot if you do SEO
`*_to_df`: a set of convenience functions for converting to DataFrames (log files, XML sitemaps, robots.txt files, and lists of URLs)

1.2 advertools Command Line Interface (CLI)

Once you install advertools with `python3 -m pip install advertools`, you should have acess to the command line interface and run the available commands.

You just need Python3 installed, and you are good to go (no need for any Python programming to use the CLI)

Run `advertools --help` or `adv -h` to get access to the documentation.

For specific documentation of a certain command run `advertools <command> --help`

For example `advertools sitemaps --help` or `adv crawl -h`

1.2.1 convert a robots.txt file (or list of file URLs) to a table in a CSV format

usage: `advertools robots [-h] [url ...]`

convert a robots.txt file (or list of file URLs) to a table in a CSV format

you can provide a web URL, or a local file URL on your local machine e.g. file:///Users/path/to/robots.txt

advertools robots https://www.google.com/robots.txt

multiple robots files:

advertools robots https://www.google.com/robots.txt https://www.google.jo/robots.txt https://www.google.es/robots.txt

use output redirection ">" to save to a CSV file:

advertools robots https://www.google.com/robots.txt > google_robots.csv

run the function for a long list of robots files saved in a text file (robotslist.txt):

advertools robots < robotslist.txt > multi_robots.csv

positional arguments: url a robots.txt URL (or a list of URLs) (default: None)

optional arguments:

-h, --help show this help message and exit

1.2.2 download, parse, and save an XML sitemap to a table in a CSV file

usage: `advertools sitemaps [-h] [-r {0,1}] [-s SEPARATOR] [sitemap_url]`

download, parse, and save an XML sitemap to a table in a CSV file

positional arguments: sitemap_url the URL of the XML sitemap (regular or sitemap index) (default: None)

optional arguments:

-h, --help show this help message and exit

-r {0,1}, --recursive {0,1} whether or not to fetch sub-sitemaps if it is a sitemap index file (default: 1)

-s SEPARATOR, --separator SEPARATOR the separator with which to separate columns of the output (default: ,)

1.2.3 split a list of URLs into their components: scheme, netloc, path, query, etc.

usage: `advertools urls [-h] [url_list ...]`

split a list of URLs into their components: scheme, netloc, path, query, etc.

positional arguments: url_list a list of URLs to parse (default: None)

optional arguments:

-h, --help show this help message and exit

1.2.4 crawl a list of known URLs using the HEAD method

usage: `advertools headers [-h] [-s [CUSTOM_SETTINGS ...]] [url_list ...] output_file`

crawl a list of known URLs using the HEAD method return status codes and all available response headers

positional arguments: url_list a list of URLs (default: None) output_file filepath - where to save the output (.jl)

optional arguments:

-h, --help show this help message and exit

-s [CUSTOM_SETTINGS ...], --custom-settings [CUSTOM_SETTINGS ...] settings that modify the behavior of the crawler settings should be separated by spaces, and each setting name and value should be separated by an equal sign '=' without spaces between them

example:

advertools headers https://example.com example.jl --custom-settings LOG_FILE=logs.log CLOSESPIDER_TIM (default: None)

1.2.5 parse, compress and convert a log file to a DataFrame in the .parquet format

usage: `advertools logs [-h] [-f [FIELDS ...]] log_file output_file errors_file log_format`

parse, compress and convert a log file to a DataFrame in the .parquet format

positional arguments: log_file filepath - the log file output_file filepath - where to save the output (.parquet) errors_file filepath - where to save the error lines (.txt) log_format the format of the logs, available defaults are:

common, combined, common_with_vhost, nginx_error, apache_error supply a special regex instead if you have a different format

optional arguments:

-h, --help show this help message and exit

-f [FIELDS ...], --fields [FIELDS ...] in case you have a special log format, provide a list of the fields names which will become column names in the parsed compressed file (default: None)

1.2.6 perform a reverse DNS lookup on a list of IP addresses

usage: `advertools dns [-h] [ip_list ...]`

perform a reverse DNS lookup on a list of IP addresses

positional arguments: ip_list a list of IP addresses (default: None)

optional arguments:

-h, --help show this help message and exit

1.2.7 generate a table of SEM keywords by supplying a list of products and a list of intent words

usage: `advertools semkw [-h] [-t [{exact,phrase,modified,broad} ...]] [-l MAX_LEN] [-c {0,1}] [-m {0,1}] [-n CAMPAIGN_NAME] products words`

generate a table of SEM keywords by supplying a list of products and a list of intent words

positional arguments: products a file containing the products that you sell, one per line words a file containing the intent words/phrases that you want to combine with products, one per line

optional arguments:

-h, --help show this help message and exit

-t [{exact,phrase,modified,broad} ...], --match-types [{exact,phrase,modified,broad} ...] -l MAX_LEN, --max-len MAX_LEN

the number of words that should be combined with products (default: 3)

-c {0,1}, --capitalize-adgroups {0,1} whether or not to capitalize ad group names in the output file (default: 1)

-m {0,1}, --order-matters {0,1} do you want combinations and permutations, or just combinations? "buy product" and "product buy" or just "buy product"? (default: 1)

-n CAMPAIGN_NAME, --campaign-name CAMPAIGN_NAME

1.2.8 get stopwords of the selected language

usage: `advertools stopwords [-h] {arabic,azerbaijani,bengali,catalan,chinese,croatian,danish,dutch,english,finnish,french,german,greek,hebrew,hindi,hungarian,indonesian,irish,italian,japanese,kazakh,nepali,norwegian,persian,polish,portuguese,romanian,russian,sinhala,spanish,swedish,tagalog,tamil,tatar,telugu,thai,turkish,ukrainian,urdu,vietnamese}`

get stopwords of the selected language

positional arguments: {arabic,azerbaijani,bengali,catalan,chinese,croatian,danish,dutch,english,finnish,french,german,greek,heb

optional arguments:

-h, --help show this help message and exit

1.2.9 get word counts of a text list optionally weighted by a number list

usage: `advertools wordfreq [-h] [-n NUMBER_LIST] [-r REGEX] [-l PHRASE_LEN] [-s [STOPWORDS ...]] [text_list ...]`

get word counts of a text list optionally weighted by a number list

words (tokens) can be tokenized using any pattern with the --regex option word/phrase lengths can also be modified using the --phrase-len option

positional arguments: text_list a text list, one document (sentence, tweet, etc.) per line (default: None)

optional arguments:

-h, --help show this help message and exit

-n NUMBER_LIST, --number-list NUMBER_LIST filepath - a file containing the number list, one number per line (default: None)

-r REGEX, --regex REGEX a regex to tokenize words (default: None)

-l PHRASE_LEN, --phrase-len PHRASE_LEN the phrase (token) length to split words (the *n* in n-grams) (default: 1)

-s [STOPWORDS ...], --stopwords [STOPWORDS ...] a list of stopwords to exclude when counting, defaults to English stopwords run *advertools stopwords english* to get the stopwords change the language to get other stopwords (default: None)

1.2.10 search for emoji using a regex

usage: `advertools emoji [-h] regex`

search for emoji using a regex

positional arguments: regex pattern to search for emoji

optional arguments:

-h, --help show this help message and exit

1.2.11 extract structured entities from a text list; emoji, hashtags, mentions

usage: `advertools extract [-h] {emoji,hashtags,mentions} text_list`

extract structured entities from a text list; emoji, hashtags, mentions

positional arguments:

{emoji,hashtags,mentions} which entity you want to extract

text_list filepath - a file containing the text list, one phrase per line

optional arguments:

-h, --help show this help message and exit

1.2.12 tokenize documents (phrases, keywords, tweets, etc) into token of the desired length

usage: `advertools tokenize [-h] [-l LENGTH] [-s SEPARATOR] [text_list ...]`

tokenize documents (phrases, keywords, tweets, etc) into token of the desired length

positional arguments: text_list filepath - a file containing the text list, one document (sentence, tweet, etc.) per line (default: None)

optional arguments:

-h, --help show this help message and exit

-l LENGTH, --length LENGTH the length of tokens (the n in n-grams) (default: 1)

-s SEPARATOR, --separator SEPARATOR the character with which to separate the tokens (default: ,)

1.2.13 SEO crawler

usage: advertools crawl [-h] [-l FOLLOW_LINKS] [-d [ALLOWED_DOMAINS ...]] [--exclude-url-params [EXCLUDE_URL_PARAMS ...]] [--include-url-params [INCLUDE_URL_PARAMS ...]] [--exclude-url-regex EXCLUDE_URL_REGEX] [--include-url-regex INCLUDE_URL_REGEX] [--css-selectors [CSS_SELECTORS ...]] [--xpath-selectors [XPATH_SELECTORS ...]] [--custom-settings [CUSTOM_SETTINGS ...]] [url_list ...] output_file

SEO crawler

positional arguments: url_list one or more URLs to crawl (default: None) output_file filepath - where to save the output (.jl)

optional arguments:

-h, --help show this help message and exit

-l FOLLOW_LINKS, --follow-links FOLLOW_LINKS whether or not to follow links encountered on crawled pages (default: 0)

-d [ALLOWED_DOMAINS ...], --allowed-domains [ALLOWED_DOMAINS ...] while following links, only links on these domains will be followed (default: None)

--exclude-url-params [EXCLUDE_URL_PARAMS ...] a list of URL parameters to exclude while following links if a link contains any of those parameters, don't follow it setting it to True will exclude links containing any parameter (default: None)

--include-url-params [INCLUDE_URL_PARAMS ...] a list of URL parameters to include while following links if a link contains any of those parameters, follow it having the same parmeters to include and exclude raises an error (default: None)

--exclude-url-regex EXCLUDE_URL_REGEX a regular expression of a URL pattern to exclude while following links if a link matches the regex don't follow it (default: None)

--include-url-regex INCLUDE_URL_REGEX a regular expression of a URL pattern to include while following links if a link matches the regex follow it (default: None)

--css-selectors [CSS_SELECTORS ...] a dictionary mapping names to CSS selectors the names will become column headers, and the selectors will be used to extract the required data/content (default: None)

--xpath-selectors [XPATH_SELECTORS ...] a dictionary mapping names to XPath selectors. the names will become column headers, and the selectors will be used to extract the required data/content (default: None)

--custom-settings [CUSTOM_SETTINGS ...] a dictionary of optional custom settings that you might want to add to the spider's functionality. there are over 170 settings for all kinds of options for details please refer to the spider settings: https://docs.scrapy.org/en/latest/topics/settings.html (default: None)

crawl a website starting from its home page:

advertools crawl https://examle.com example_output.jl --follow-links 1

crawl a list of pages (list mode):

advertools crawl url_1 url_2 url_3 example_output.jl

OR if you have a long list in a file (url_list.txt):

advertools crawl < url_list.txt example_output.jl

stop crawling after having processed 1,000 pages:

advertools crawl https://examle.com example_output.jl --follow-links 1 --custom-settings CLOSESPIDER_PAGECOUNT=1000

To install advertools, run the following from the command line:

```
pip install advertools
# OR:
pip3 install advertools
```

1.3 Generate Keywords for SEM Campaigns

A big part of setting up SEM campaigns consists of generating keywords, and properly mapping them to landing pages and ads, as well as putting them in the right campaign and ad group structure.

Keyword research is the part of this task that takes the most time. It is very tedious, yet extremely important.

The shift here is that we are going to be *generating* keywords as opposed to researching them.

What is a keyword anyway?

It is basically a phrase that contains two things:

Product This is the thing that you are selling. It is simply the name of it. "barcelona", "guitar", "rio de janeiro", "accounting". The product on its own is not enough for us to understand what the user is looking for. "barcelona trips" and "barcelona football club" are completely different "keywords" for example.

Word To give meaning to the product, it has to come with a word. The word can be a verb like "buy" or "purchase", and it can also be another noun, but with a clear intent expressed; "price" and "offers" for example clearly show purchase intent.

So, to *generate* keywords we need phrases that contain both, the product and the descriptive word(s). It is very easy to get the products as you know what you sell. The next thing you need to come up with are the words that work within your strategy. The most import idea here is that once you determine that you sell courses for example, there aren't really that many words that can describe that intent; course, courses, tutorial, certification, learn, learning, education, etc. How many can you come up with? How many exist in any language? Fifteen, twenty? Once you have those are basically done.

Depending on what service you provide and what segment of the market you target it shouldn't be difficult to come up with ideas for words (not keywords yet). You might have an e-commerce site, but want to mainly focus on cheap and discounted products. Or maybe you have luxury items, and want to exclude words that signify price sensitivity.

Let's say you have a job site and you know that you provide jobs for engineering, graphic design, and marketing. The words are easy to come up with; "job", "jobs", "careers", "vacancies", "full time", "part time", "work", and so on.

Now what we can do is use the *kw_generate* function to come up with all possible combinations (order doesn't matter) and/or permutations (order matters) and get a ready-to-use table to upload and start running the campaign.

```
import advertools as adv

products = ['enginering', 'graphic design', 'marketing']
words = ['jobs', 'careers', 'vacancies', 'full time', 'part time']

kw_df = adv.kw_generate(products, words)
kw_df
```

```
         Campaign    Ad Group                          Keyword    Criterion Type  ␣
↪         Labels
0    SEM_Campaign  Enginering                enginering jobs            Exact  ␣
↪           Jobs
1    SEM_Campaign  Enginering                enginering jobs           Phrase  ␣
↪           Jobs
2    SEM_Campaign  Enginering                +enginering +jobs            Broad  ␣
↪           Jobs
3    SEM_Campaign  Enginering              enginering careers            Exact  ␣
↪        Careers
4    SEM_Campaign  Enginering              enginering careers           Phrase  ␣
↪        Careers
..            ...         ...                             ...              ...  ␣
↪           ...
625  SEM_Campaign   Marketing       part time vacancies marketing           Phrase  ␣
↪Part Time;Vacancies
626  SEM_Campaign   Marketing   +part +time +vacancies +marketing            Broad  ␣
↪Part Time;Vacancies
627  SEM_Campaign   Marketing       part time full time marketing            Exact  ␣
↪Part Time;Full Time
628  SEM_Campaign   Marketing       part time full time marketing           Phrase  ␣
↪Part Time;Full Time
629  SEM_Campaign   Marketing  +part +time +full +time +marketing            Broad  ␣
↪Part Time;Full Time
[630 rows x 5 columns]
```

Check the *kw_generate()* function for more options and details. Once you have your keywords done, you can start creating ads using either the *ad_create* function (bottom-up approach) or the *ad_from_string* function (top-down approach).

kw_broad(*words*)

Return `words` in broad match.

Parameters **words** (*list*) -- list of strings

Returns **formatted** `words` in broad match type

```
>>> keywords = ['[learn guitar]', '"guitar courses"', '+guitar +tutor']
>>> kw_broad(keywords)
['learn guitar', 'guitar courses', 'guitar tutor']
```

kw_exact(*words*)

Return words in exact match.

Parameters **words** (*list*) -- list of strings

Returns formatted words in exact match type

```
>>> keywords = ['learn guitar', 'guitar courses', 'guitar tutor']
>>> kw_exact(keywords)
['[learn guitar]', '[guitar courses]', '[guitar tutor]']
```

kw_generate(*products*, *words*, *max_len=3*, *match_types=('Exact', 'Phrase', 'Modified')*, *capitalize_adgroups=True*, *order_matters=True*, *campaign_name='SEM_Campaign'*)

Generate a data frame of keywords using a list of products and relevant words.

Parameters

- **products** (*list*) -- will be used as the names of the ad groups
- **words** (*list*) -- related words that make it clear that the user is interested in products
- **max_len** (*int*) -- the maximum number of words to include in each permutation of final keywords
- **match_types** (*list*) -- one or more of ('Exact', 'Phrase', 'Modified', 'Broad')
- **capitalize_adgroups** (*bool*) -- whether or not to set adgroup names in the "Ad Group" column to title case or keep them as is, default True
- **order_matters** (*bool*) -- whether or not the order of words in keywords matters, default False
- **campaign_name** (*str*) -- name of campaign

Returns keywords_df a pandas.DataFrame ready to upload

```
>>> import advertools as adv
>>> products = ['bmw', 'toyota']
>>> words = ['buy', 'second hand']
>>> kw_df = adv.kw_generate(products, words)
>>> kw_df.head()
       Campaign Ad Group          Keyword Criterion Type       Labels
0  SEM_Campaign      Bmw          bmw buy          Exact          Buy
1  SEM_Campaign      Bmw          bmw buy         Phrase          Buy
2  SEM_Campaign      Bmw        +bmw +buy          Broad          Buy
3  SEM_Campaign      Bmw  bmw second hand          Exact  Second Hand
4  SEM_Campaign      Bmw  bmw second hand         Phrase  Second Hand
```

```
>>> kw_df.tail()
        Campaign Ad Group                    Keyword Criterion Type           Labels
55  SEM_Campaign   Toyota     second hand toyota buy         Phrase  Second Hand;Buy
56  SEM_Campaign   Toyota  +second hand +toyota +buy          Broad  Second Hand;Buy
57  SEM_Campaign   Toyota     second hand buy toyota          Exact  Second Hand;Buy
```

(continues on next page)

(continued from previous page)

```
58  SEM_Campaign   Toyota     second hand buy toyota         Phrase  Second Hand;Buy
59  SEM_Campaign   Toyota  +second hand +buy +toyota          Broad  Second Hand;Buy
```

Sometimes you want to retain capitalization and keep it as it as is in the "Ad Group" column. This is especially important for consistency with ads DataFrames for easier integration between the two. Set *capitalize_adgroups=False* to keep capitalization the same:

```
>>> adv.kw_generate(['SEO'], ['services', 'provider'], capitalize_adgroups=False).
↪head()
       Campaign Ad Group         Keyword Criterion Type    Labels
0  SEM_Campaign      SEO    SEO services          Exact  Services
1  SEM_Campaign      SEO    SEO services         Phrase  Services
2  SEM_Campaign      SEO  +SEO +services          Broad  Services
3  SEM_Campaign      SEO    SEO provider          Exact  Provider
4  SEM_Campaign      SEO    SEO provider         Phrase  Provider
```

kw_modified(*words*)

Return `words` in modified broad match.

Parameters **words** (*list*) -- list of strings

Returns formatted `words` in modified broad match type

```
>>> keywords = ['learn guitar', 'guitar courses', 'guitar tutor']
>>> kw_modified(keywords)
['+learn +guitar', '+guitar +courses', '+guitar +tutor']
```

kw_neg_broad(*words*)

Return `words` in negative broad match.

Parameters **words** (*list*) -- list of strings

Returns formatted `words` in negative broad match type

```
>>> keywords = ['learn guitar', 'guitar courses', 'guitar tutor']
>>> kw_neg_broad(keywords)
['-learn guitar', '-guitar courses', '-guitar tutor']
```

kw_neg_exact(*words*)

Return `words` in negative exact match.

Parameters **words** (*list*) -- list of strings

Returns formatted `words` in negative exact match type

```
>>> keywords = ['learn guitar', 'guitar courses', 'guitar tutor']
>>> kw_neg_exact(keywords)
['-[learn guitar]', '-[guitar courses]', '-[guitar tutor]']
```

kw_neg_phrase(*words*)

Return `words` in negative phrase match.

Parameters **words** (*list*) -- list of strings

Returns formatted `words` in negative phrase match type

```
>>> keywords = ['learn guitar', 'guitar courses', 'guitar tutor']
>>> kw_neg_phrase(keywords)
['-"learn guitar"', '-"guitar courses"', '-"guitar tutor"']
```

kw_phrase(*words*)

Return words in phrase match.

Parameters **words** (*list*) -- list of strings

Returns formatted words in phrase match type

```
>>> keywords = ['learn guitar', 'guitar courses', 'guitar tutor']
>>> kw_phrase(keywords)
['"learn guitar"', '"guitar courses"', '"guitar tutor"']
```

1.4 Create Ads on a Large Scale

When creating large-scale campaigns, you also need to create ads on a large scale. For products in a similar category you typically want to use the same ads, but only replace the product name, "Get the latest <product> now", and replace *product* as many times as you have ads.

```
import advertools as adv

products = ['Dubai', 'Tokyo', 'Singapore']
adv.ad_create(template='5-star Hotels in {}',
              replacements=products,
              max_len=30,
              fallback='Great Cities')
```

```
['5-star Hotels In Dubai',
'5-star Hotels In Tokyo',
'5-star Hotels In Singapore']
```

An important thing to to watch out for, is long product names. Since text ads have limits on each slot, you need to make sure you don't exceed that limit. For this you need to provide a *fallback* text in case the product name is longer than *max_len*.

```
products = ['Lisbon', 'Porto', 'Algarve', 'Freixo de Espada à Cinta']
adv.ad_create(template='5-star Hotels in {}',
              replacements=products,
              max_len=30,
              fallback='Portugal')
```

```
['5-star Hotels In Lisbon',
'5-star Hotels In Porto',
'5-star Hotels In Algarve',
'5-star Hotels In Portugal']
```

ad_create(*template*, *replacements*, *fallback*, *max_len=30*, *capitalize=True*)

Insert each of the replacement strings in its place within template.

Parameters

- **template** (*str*) -- a string format template, using braces e.g. "Get the latest {} today."
- **replacements** (*list*) -- replacement strings to be inserted in `template`
- **fallback** (*str*) -- the string to insert in `template` in case `replacement` is longer than `max_len`
- **max_len** (*int*) -- the maximum allowed length of the full string
- **capitalize** (*bool*) -- whether or not to capitalize words in the result

Returns formatted list of strings

```
>>> ad_create("Let's count {}", ['one', 'two', 'three'], 'one', 20)
["Let's Count One", "Let's Count Two", "Let's Count Three"]
```

```
>>> ad_create(template='My favorite car is {}',
...           replacements=['Toyota', 'BMW', 'Mercedes', 'Lamborghini'],
...           fallback='great',
...           max_len=28)
['My Favorite Car Is Toyota', 'My Favorite Car Is Bmw',
'My Favorite Car Is Mercedes', 'My Favorite Car Is Great']
```

```
>>> ad_create('KeEP cApITalization {}', ['As IS'],
...           fallback='fallback', max_len=50, capitalize=False)
['KeEP cApITalization As IS']
```

```
>>> ad_create('This is very long and will produce and error',
...           replacements=['something', 'long'], fallback='Very long',
...           max_len=20)
Traceback (most recent call last):
File "<input>", line 1, in <module>
File "<input>", line 26, in ad_create
ValueError: template + fallback should be <= 20 chars
```

1.5 Create Ads Using Long Descriptive Text (top-down approach)

Many times you have long descriptive text about your products, especially on their respective landing pages. The allowed length of text ads has become considerably long on many platforms. On Google Ads for example, you have slots of 30, 30, 30, 90, and 90 characters, for a total of 270. That's more than enough space to talk about the main features of your product.

The question is, how do you utilize that long description text that has all the details that you want, and make sure it fits correctly within the limits given by the platform you are using?

The *ad_from_string()* function does exactly that. Given a long string, it divides it into slots of any given length that you specify, and if any text remains it will be appended to the end of the returned list.

Another important benefit of this is that you can take those long descriptions (or write them) once, and then you can easily split them into different slots based on the ad format and the platform you are using.

Here is a quick overview of the available parameters and options:

s The string that you want to split. This would typically be available on the landing pages of each product.

slots The lengths that you want to split into. Note that although the default uses Google Ads' text ad template, you can change it to any other group of slots, with more or fewer slots of different lengths.

sep The separator by which to split the text. The default is `None` which splits the text by whitespace, but you can change it to something else if needed. Sometimes you might want the text split by hyphens (URLs for example) so you can split by that character.

capitalize The default is `False` which leaves the capitalization of `s` intact. If you set it to `True` then the first letter of each word would be capitalized.

Example

Note that in any case, the returned list of characters is longer than the provided slots by one. So if you provide five slots, for example, the function will always return a list of length six.

This is to ensure that the remainder of the text is not lost if it is longer, so you know what is missing. In case you have shorter text, you will still have one element more than the provided slots to ensure consistency.

```
import advertools as adv
desc_text = "Get the latest gadget online. The GX12 model comes with 13 things that do a␣
↪lot of good stuff for your health. Start shopping now."
len(desc_text)  # 130
```

Now let's see how this same description can be utilized in different scenarios

1.5.1 Google Text Ads

Since this is shorter than the default Google values, you will get extra empty slots (with an additional last one).

```
adv.ad_from_string(desc_text)  # default values (Google text ads)
```

```
['Get the latest gadget online.',
'The GX12 model comes with 13',
'things that do a lot of good',
'stuff for your health. Start shopping now.',
'',
'',
'',
'']
```

1.5.2 Facebook Feed Ads

In this case, it is also shorter than the default value, so you get an extra space.

```
adv.ad_from_string(desc_text, [125, 25, 30])  # Facebook feed ads
```

Since it might not look good having just one word in the second slot, and an empty last one, you might want to change it as follows:

```
adv.ad_from_string(desc_text, [90, 25, 30])
```

```
['Get the latest gadget online. The GX12 model comes with 13 things that do a lot of good
↪',
'stuff for your health.',
'Start shopping now.',
'']
```

1.5.3 Facebook Instant Article Ad

Here is a case where our text is longer than the provided limitations, so we end up having an extra space that is not used:

```
adv.ad_from_string(desc_text, [25, 30])  # Facebook instant article ad
```

```
['Get the latest gadget',
'online. The GX12 model comes',
'with 13 things that do a lot of good stuff for your health. Start shopping now.']
```

ad_from_string(*s*, *slots=(30, 30, 30, 90, 90, 15, 15)*, *sep=None*, *capitalize=False*)

Convert string `s` to an ad by splitting it into groups of words. Each group would have a length of at most the allowed length for that slot.

If the total length of `s` exceeds the total allowed length, all remaining characters would be grouped in the last element of the returned list.

Parameters

- **s** (*str*) -- a string of characters, with no restrictions on length
- **slots** (*list*) -- an iterable of integers for the maximum lengths for each slot
- **sep** (*str*) -- character(s) by which to split `s`
- **capitalize** (*bool*) -- whether or not to capitalize each word after grouping. Setting it as False would not change the capitalization of the input string

Returns text_ad a list of strings

```
>>> ad_from_string('this is a short ad')
['this is a short ad', '', '', '', '', '', '', '']
```

```
>>> ad_from_string('this is a longer ad and will take the first two slots')
['this as a longer ad and would', 'take the first two slots',
'', '', '', '', '', '']
```

```
>>> ad_from_string("Slots can be changed the way you want", (10, 15, 10))
['Slots can', 'be changed the', 'way you', 'want']
```

```
>>> ad_from_string("The capitalization REMAinS as IS bY DefAULt",
...                        (10, 15, 10))
['The', 'capitalization', 'REMAinS as', 'IS bY DefAULt']
```

```
>>> ad_from_string("set captialize=True to capitalize first letters",
...                capitalize=True)
['Set Captialize=true To', 'Capitalize First Letters',
 '', '', '', '', '', '']
```

1.6 Analyze and Test robots.txt Files on a Large Scale

Even though they are tiny in size, robots.txt files contain potent instructions that can block major sections of your site, which is what they are supposed to do. Only sometimes you might make the mistake of blocking the wrong section.

So it is very important to check if certain pages (or groups of pages) are blocked for a certain user-agent by a certain robots.txt file. Ideally, you would want to run the same check for all possible user-agents. Even more ideally, you want to be able to run the check for a large number of pages with every possible combination with user-agents.

To get the robots.txt file into an easily readable format, you can use the *robotstxt_to_df()* function to get it in a DataFrame.

```
import advertools as adv

amazon = adv.robotstxt_to_df('https://www.amazon.com/robots.txt')
amazon
```

	directive	content	etag	robotstxt_last_modified	robotstxt_url	download_date
0	User-agent	*	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
1	Disallow	/exec/obidos/account-access-login	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
2	Disallow	/exec/obidos/change-style	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
3	Disallow	/exec/obidos/flex-sign-in	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
4	Disallow	/exec/obidos/handle-buy-box	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
...	...	...	...	...	...	...
146	Disallow	/hp/video/mystuff	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
147	Disallow	/gp/video/profiles	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
148	Disallow	/hp/video/profiles	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
149	User-agent	EtaoSpider	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00
150	Disallow	/	"a850165d925db701982[illegible]92d3"	2022-02-11 17:51:39+00:00	https://www.amazon.com/robots.txt	2022-02-11 19:33:03.200689+00:00

The returned DataFrame contains columns for directives, their content, the URL of the robots.txt file, as well as the

date it was downloaded.

- *directive*: The main commands. Allow, Disallow, Sitemap, Crawl-delay, User-agent, and so on.
- *content*: The details of each of the directives.
- *robotstxt_last_modified*: The date when the robots.txt file was last modified, if provided (according the response header Last-modified).
- *etag*: The entity tag of the response header, if provided.
- *robotstxt_url*: The URL of the robots.txt file.
- *download_date*: The date and time when the file was downloaded.

Alternatively, you can provide a list of robots URLs if you want to download them all in one go. This might be interesting if:

- You are analyzing an industry and want to keep an eye on many different websites.
- You are analyzing a website with many sub-domains, and want to get all the robots files together.
- You are trying to understand a company that has many websites under different domains and sub-domains.

In this case you simply provide a list of URLs instead of a single one.

```
robots_urls = ['https://www.google.com/robots.txt',
               'https://twitter.com/robots.txt',
               'https://facebook.com/robots.txt']

googtwfb = adv.robotstxt_to_df(robots_urls)

# How many lines does each robots file have?
googtwfb.groupby('robotstxt_url')['directive'].count()
```

```
robotstxt_url
https://facebook.com/robots.txt      541
https://twitter.com/robots.txt       108
https://www.google.com/robots.txt    289
Name: directive, dtype: int64
```

```
# Display the first five rows of each of the robots files:
googtwfb.groupby('robotstxt_url').head()
```

	directive	content	robotstxt_last_modified	robotstxt_url	download_date
0	User-agent	*	2022-02-07 22:30:00+00:00	https://www.google.com/robots.txt	2022-02-11 19:52:13.375724+00:00
1	Disallow	/search	2022-02-07 22:30:00+00:00	https://www.google.com/robots.txt	2022-02-11 19:52:13.375724+00:00
2	Allow	/search/about	2022-02-07 22:30:00+00:00	https://www.google.com/robots.txt	2022-02-11 19:52:13.375724+00:00
3	Allow	/search/static	2022-02-07 22:30:00+00:00	https://www.google.com/robots.txt	2022-02-11 19:52:13.375724+00:00
4	Allow	/search/howsearchworks	2022-02-07 22:30:00+00:00	https://www.google.com/robots.txt	2022-02-11 19:52:13.375724+00:00
289	comment	Google Search Engine Robot	NaT	https://twitter.com/robots.txt	2022-02-11 19:52:13.461815+00:00
290	comment		NaT	https://twitter.com/robots.txt	2022-02-11 19:52:13.461815+00:00
291	User-agent	Googlebot	NaT	https://twitter.com/robots.txt	2022-02-11 19:52:13.461815+00:00
292	Allow	/?_escaped_fragment_	NaT	https://twitter.com/robots.txt	2022-02-11 19:52:13.461815+00:00
293	Allow	/*?lang=	NaT	https://twitter.com/robots.txt	2022-02-11 19:52:13.461815+00:00
397	comment	Notice: Collection of data on Facebook through automated means is	NaT	https://facebook.com/robots.txt	2022-02-11 19:52:13.474456+00:00
398	comment	prohibited unless you have express written permission from Facebook	NaT	https://facebook.com/robots.txt	2022-02-11 19:52:13.474456+00:00
399	comment	and may only be conducted for the limited purpose contained in said	NaT	https://facebook.com/robots.txt	2022-02-11 19:52:13.474456+00:00
400	comment	permission.	NaT	https://facebook.com/robots.txt	2022-02-11 19:52:13.474456+00:00
401	comment	See: http://www.facebook.com/apps/site_scraping_tos_terms.php	NaT	https://facebook.com/robots.txt	2022-02-11 19:52:13.474456+00:00

1.6.1 Bulk `robots.txt` Tester

This tester is designed to work on a large scale. The *`robotstxt_test()`* function runs a test for a given robots.txt file, checking which of the provided user-agents can fetch which of the provided URLs, paths, or patterns.

```
robotstxt_test('https://www.example.com/robots.txt',
               useragents=['Googlebot', 'baiduspider', 'Bingbot']
               urls=['/', '/hello', '/some-page.html']])
```

As a result, you get a DataFrame with a row for each combination of (user-agent, URL) indicating whether or not that particular user-agent can fetch the given URL.

Some reasons why you might want to do that:

- SEO Audits: Especially for large websites with many URL patterns, and many rules for different user-agents.
- Developer or site owner about to make large changes
- Interest in strategies of certain companies

1.6.2 User-agents

In reality there are only two groups of user-agents that you need to worry about:

- User-agents listed in the robots.txt file: For each one of those you need to check whether or not they are blocked from fetching a certain URL (or pattern).
- * all other user-agents: The * includes all other user-agents, so checking the rules that apply to it should take care of the rest.

1.6.3 robots.txt Testing Approach

1. Get the robots.txt file that you are interested in
2. Extract the user-agents from it
3. Specify the URLs you are interested in testing
4. Run the *robotstxt_test()* function

```
fb_robots = adv.robotstxt_to_df('https://www.facebook.com/robots.txt')
fb_robots
```

	directive	content	robotstxt_url	download_date
0	comment	Notice: Collection of data on Facebook through automated means is	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
1	comment	prohibited unless you have express written permission from Facebook	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
2	comment	and may only be conducted for the limited purpose contained in said	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
3	comment	permission.	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
4	comment	See: http://www.facebook.com/apps/site_scraping_tos_terms.php	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
...	...	...	...	...
536	Allow	/ajax/pagelet/generic.php/PagePostsSectionPagelet	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
537	Allow	/careers/	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
538	Allow	/safetycheck/	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
539	User-agent	*	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00
540	Disallow	/	https://www.facebook.com/robots.txt	2022-02-12 00:48:58.951053+00:00

Now that we have downloaded the file, we can easily extract the list of user-agents that it contains.

```
fb_useragents = (fb_robots
                 [fb_robots['directive']=='User-agent']
                 ['content'].drop_duplicates()
                .tolist())
fb_useragents
```

```
['Applebot',
 'baiduspider',
 'Bingbot',
 'Discordbot',
 'facebookexternalhit',
 'Googlebot',
 'Googlebot-Image',
```

(continues on next page)

(continued from previous page)

```
 'ia_archiver',
 'LinkedInBot',
 'msnbot',
 'Naverbot',
 'Pinterestbot',
 'seznambot',
 'Slurp',
 'teoma',
 'TelegramBot',
 'Twitterbot',
 'Yandex',
 'Yeti',
 '*']
```

Quite a long list!

As a small and quick test, I'm interested in checking the home page, a random profile page (/bbc), groups and hashtags pages.

```
urls_to_test = ['/', '/bbc', '/groups', '/hashtag/']
fb_test = robotstxt_test('https://www.facebook.com/robots.txt',
                         fb_useragents, urls_to_test)
fb_test
```

	robotstxt_url	user_agent	url_path	can_fetch
0	https://www.facebook.com/robots.txt	*	/	False
1	https://www.facebook.com/robots.txt	*	/bbc	False
2	https://www.facebook.com/robots.txt	*	/groups	False
3	https://www.facebook.com/robots.txt	*	/hashtag/	False
4	https://www.facebook.com/robots.txt	Applebot	/	True
	...	...	...	...
75	https://www.facebook.com/robots.txt	seznambot	/hashtag/	True
76	https://www.facebook.com/robots.txt	teoma	/	True
77	https://www.facebook.com/robots.txt	teoma	/bbc	True
78	https://www.facebook.com/robots.txt	teoma	/groups	True
79	https://www.facebook.com/robots.txt	teoma	/hashtag/	True

For twenty user-agents and four URLs each, we received a total of eighty test results. You can immediately see that all user-agents not listed (denoted by * are not allowed to fetch any of the provided URLs).

Let's see who is and who is not allowed to fetch the home page.

```
fb_test.query('url_path== "/"')
```

	robotstxt_url	user_agent	url_path	can_fetch
0	https://www.facebook.com/robots.txt	*	/	False
4	https://www.facebook.com/robots.txt	Applebot	/	True
8	https://www.facebook.com/robots.txt	Bingbot	/	True
12	https://www.facebook.com/robots.txt	Discordbot	/	False
16	https://www.facebook.com/robots.txt	Googlebot	/	True
20	https://www.facebook.com/robots.txt	Googlebot-Image	/	True
24	https://www.facebook.com/robots.txt	LinkedInBot	/	False
28	https://www.facebook.com/robots.txt	Naverbot	/	True
32	https://www.facebook.com/robots.txt	Pinterestbot	/	False
36	https://www.facebook.com/robots.txt	Slurp	/	True
40	https://www.facebook.com/robots.txt	TelegramBot	/	False
44	https://www.facebook.com/robots.txt	Twitterbot	/	True
48	https://www.facebook.com/robots.txt	Yandex	/	True
52	https://www.facebook.com/robots.txt	Yeti	/	True
56	https://www.facebook.com/robots.txt	baiduspider	/	True
60	https://www.facebook.com/robots.txt	facebookexternalhit	/	False
64	https://www.facebook.com/robots.txt	ia_archiver	/	False
68	https://www.facebook.com/robots.txt	msnbot	/	True
72	https://www.facebook.com/robots.txt	seznambot	/	True
76	https://www.facebook.com/robots.txt	teoma	/	True

I'll leave it to you to figure out why LinkedIn and Pinterest are not allowed to crawl the home page but Google and Apple are, because I have no clue!

robotstxt_test(*robotstxt_url*, *user_agents*, *urls*)

Given a `robotstxt_url` check which of the `user_agents` is allowed to fetch which of the `urls`.

All the combinations of `user_agents` and `urls` will be checked and the results returned in one DataFrame.

```
>>> robotstxt_test('https://facebook.com/robots.txt',
...                user_agents=['*', 'Googlebot', 'Applebot'],
...                urls=['/', '/bbc', '/groups', '/hashtag/'])
                   robotstxt_url user_agent   url_path  can_fetch
0   https://facebook.com/robots.txt          *          /      False
1   https://facebook.com/robots.txt          *       /bbc      False
2   https://facebook.com/robots.txt          *    /groups      False
3   https://facebook.com/robots.txt          *  /hashtag/      False
4   https://facebook.com/robots.txt   Applebot          /       True
5   https://facebook.com/robots.txt   Applebot       /bbc       True
6   https://facebook.com/robots.txt   Applebot    /groups       True
7   https://facebook.com/robots.txt   Applebot  /hashtag/      False
8   https://facebook.com/robots.txt  Googlebot          /       True
9   https://facebook.com/robots.txt  Googlebot       /bbc       True
10  https://facebook.com/robots.txt  Googlebot    /groups       True
11  https://facebook.com/robots.txt  Googlebot  /hashtag/      False
```

Parameters

- **robotstxt_url** (*url*) -- The URL of robotx.txt file
- **user_agents** (*str,list*) -- One or more user agents
- **urls** (*str,list*) -- One or more paths (relative) or URLs (absolute) to check

Return DataFrame robotstxt_test_df

robotstxt_to_df(*robotstxt_url*, *output_file=None*)

Download the contents of `robotstxt_url` into a DataFrame

You can also use it to download multiple robots files by passing a list of URLs.

```
>>> robotstxt_to_df('https://www.twitter.com/robots.txt')
     directive content                       robotstxt_url                  ␣
↪download_date
0   User-agent           *  https://www.twitter.com/robots.txt      2020-09-27␣
↪21:57:23.702814+00:00
1     Disallow           /  https://www.twitter.com/robots.txt      2020-09-27␣
↪21:57:23.702814+00:00
```

```
>>> robotstxt_to_df(['https://www.google.com/robots.txt',
...                  'https://www.twitter.com/robots.txt'])
       directive                                    content        robotstxt_last_
↪modified                            robotstxt_url                          ␣
↪download_date
0     User-agent                                        *      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
1       Disallow                                   /search      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
2          Allow                             /search/about      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
3          Allow                           /search/static      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
4          Allow                  /search/howsearchworks      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
283   User-agent                       facebookexternalhit      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
284        Allow                                   /imgres      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
285      Sitemap      https://www.google.com/sitemap.xml      2021-01-11␣
↪21:00:00+00:00        https://www.google.com/robots.txt      2021-01-16 14:08:50.
↪087985+00:00
286   User-agent                                        *                        ␣
↪NaT       https://www.twitter.com/robots.txt      2021-01-16 14:08:50.468588+00:00
287     Disallow                                        /                        ␣
↪NaT       https://www.twitter.com/robots.txt      2021-01-16 14:08:50.468588+00:00
```

For research purposes and if you want to download more than ~500 files, you might want to use `output_file` to save results as they are downloaded. The file extension should be ".jl", and robots files are appended to that file as soon as they are downloaded, in case you lose your connection, or maybe your patience!

```
>>> robotstxt_to_df(['https://example.com/robots.txt',
...                  'https://example.com/robots.txt',
...                  'https://example.com/robots.txt'],
...                 output_file='robots_output_file.jl')
```

To open the file as a DataFrame:

```
>>> import pandas as pd
>>> robotsfiles_df = pd.read_json('robots_output_file.jl', lines=True)
```

Parameters

- **robotstxt_url** (*url*) -- One or more URLs of the robots.txt file(s)
- **output_file** (*str*) -- Optional file path to save the robots.txt files, mainly useful for downloading > 500 files. The files are appended as soon as they are downloaded. Only ".jl" extensions are supported.

Returns DataFrame robotstxt_df A DataFrame containing directives, their content, the URL and time of download

1.7 Download, Parse, and Analyze XML Sitemaps

One of the fastest and easiest ways to get insights on a website's content is to simply download its XML sitemap(s).

Sitemaps are also important SEO tools as they reveal a lot of information about the website, and help search engines in indexing those pages. You might want to run an SEO audit and check if the URLs in the sitemap properly correspond to the actual URLs of the site, so this would be an easy way to get them.

Sitemaps basically contain a log of publishing activity, and if they have rich URLs then you can do some good analysis on their content over time as well.

The *sitemap_to_df()* function is very simple to use, and only requires the URL of a sitemap, a sitemap index, or even a robots.txt file. It goes through the sitemap(s) and returns a DataFrame containing all the tags and their information.

- *loc*: The location of the URLs of hte sitemaps.
- *lastmod*: The datetime of the date when each URL was last modified, if available.
- *sitemap*: The URL of the sitemap from which the URL on this row was retreived.
- *etag*: The entity tag of the response header, if provided.
- *sitemap_last_modified*: The datetime when the sitemap file was last modified, if provided.
- *sitemap_size_mb*: The size of the sitemap in mega bytes (1MB = 1,024 x 1,024 bytes)
- *download_date*: The datetime when the sitemap was downloaded.

1.7.1 Sitemap Index

Large websites typically have a sitmeapindex file, which contains links to all other regular sitemaps that belong to the site. The *sitemap_to_df()* function retreives all sub-sitemaps recursively by default. In some cases, especially with very large sites, it might be better to first get the sitemap index, explore its structure, and then decide which sitemaps you want to get, or if you want them all. Even with smaller websites, it still might be interesting to get the index only and see how it is structured.

This behavior can be modified by the `recursive` parameter, which is set to *True* by default. Set it to *False* if you want only the index file.

Another interesting thing you might want to do is to provide a robots.txt URL, and set *recursive=False* to get all available sitemap index files.

```
>>> sitemap_to_df('https://example.com/robots.txt', recursive=False)
```

Let's now go through a quick example of what can be done with sitemaps. We can start by getting one of the BBC's sitemaps.

1.7.2 Regular XML Sitemaps

```
import advertools as adv

bbc_sitemap = adv.sitemap_to_df('https://www.bbc.com/sitemaps/https-sitemap-com-archive-
↪1.xml')
bbc_sitemap.head(10)
```

	loc	last-mod	sitemap	etag	sitemap_last_modified	sitemap_size_mb	download_date
0	https://www.bbc.com/arabic/middleeast/2009/06/090620_as_iraq_explosion_tc2	2009-06-20 14:10:48+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
1	https://www.bbc.com/arabic/middleeast/2009/06/090620_iraq_blast_tc2	2009-06-20 21:07:43+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
2	https://www.bbc.com/arabic/business/2009/06/090622_me_worldbank_tc2	2009-06-22 12:41:48+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
3	https://www.bbc.com/arabic/multimedia/2009/06/090624_me_inpictures_brazil_tc2	2009-06-24 15:27:24+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
4	https://www.bbc.com/arabic/business/2009/06/090618_tomtest	2009-06-18 15:32:54+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
5	https://www.bbc.com/arabic/multimedia/2009/06/090625_sf_tamim_verdict_tc2	2009-06-25 09:46:39+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
6	https://www.bbc.com/arabic/middleeast/2009/06/090623_iz_cairo_russia_tc2	2009-06-23 13:10:56+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
7	https://www.bbc.com/arabic/sports/2009/06/090622_me_egypt_us_tc2	2009-06-22 15:37:07+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
8	https://www.bbc.com/arabic/sports/2009/06/090624_mz_wimbledon_tc2	2009-06-24 13:57:18+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00
9	https://www.bbc.com/arabic/worldnews/2009/06/090623_mz_leaders_lifespan_tc2	2009-06-23 13:24:23+00:00	https://www.bbc.com/sitemaps/https-sitemap-com-archive-1.xml	e7e15811c65f4061f89f89fe7163df29f5	2022-11-05 20:52:56+00:00	7.6	2022-02-12 01:37:39.461037+00:00

```
print(bbc_sitemap.shape)
print(bbc_sitemap.dtypes)
```

```
(49999, 7)

loc                                object
lastmod                datetime64[ns, UTC]
sitemap                            object
```

(continues on next page)

(continued from previous page)

```
etag                                object
sitemap_last_modified    datetime64[ns, UTC]
sitemap_size_mb                      float64
download_date            datetime64[ns, UTC]
dtype: object
```

Since `lastmod` is a `datetime` object, we can easily use it for various time-related operations. Here we look at how many articles have been published (last modified) per year.

```
bbc_sitemap.set_index('lastmod').resample('A')['loc'].count()
```

```
lastmod
2008-12-31 00:00:00+00:00     2287
2009-12-31 00:00:00+00:00    47603
2010-12-31 00:00:00+00:00        0
2011-12-31 00:00:00+00:00        0
2012-12-31 00:00:00+00:00        0
2013-12-31 00:00:00+00:00        0
2014-12-31 00:00:00+00:00        0
2015-12-31 00:00:00+00:00        0
2016-12-31 00:00:00+00:00        0
2017-12-31 00:00:00+00:00        0
2018-12-31 00:00:00+00:00        0
2019-12-31 00:00:00+00:00       99
2020-12-31 00:00:00+00:00       10
Freq: A-DEC, Name: loc, dtype: int64
```

As the majority are in 2009 with a few in other years, it seems these were later updated, but we would have to check to verify (in this special case BBC's URLs contain date information, which can be compared to `lastmod` to check if there is a difference between them).

We can take a look at a sample of the URLs to get the URL template that they use.

```
bbc_sitemap['loc'].sample(10).tolist()
```

```
['https://www.bbc.com/russian/rolling_news/2009/06/090628_rn_pakistani_soldiries_ambush',
'https://www.bbc.com/urdu/pakistan/2009/04/090421_mqm_speaks_rza',
'https://www.bbc.com/arabic/middleeast/2009/07/090723_ae_silwan_tc2',
'https://www.bbc.com/portuguese/noticias/2009/07/090729_iraquerefenbritsfn',
'https://www.bbc.com/portuguese/noticias/2009/06/090623_egitomilitaresfn',
'https://www.bbc.com/portuguese/noticias/2009/03/090302_gazaconferenciaml',
'https://www.bbc.com/portuguese/noticias/2009/07/090715_hillary_iran_cq',
'https://www.bbc.com/vietnamese/culture/2009/04/090409_machienhuu_revisiting',
'https://www.bbc.com/portuguese/noticias/2009/05/090524_paquistaoupdateg',
'https://www.bbc.com/arabic/worldnews/2009/06/090629_om_pakistan_report_tc2']
```

It seems the pattern is

https://www.bbc.com/{language}/{topic}/{YYYY}/{MM}/{YYMMDD_article_title}

This is quite a rich structure, full of useful information. We can *analyze the URL structure* using the `url_to_df` function:

```
url_df = adv.url_to_df(bbc_sitemap['loc'])
url_df
```

	url	scheme	net-loc	path	query	frag-ment	dir_1	dir_2	dir_3	dir_4	dir_5	dir_6	dir_7	last_dir
0	https://www.bbc.com/arabic/middleeast/2009/06/090620_as_iraq_explosion_tc2	https	www.bbc.com	/arabic/middleeast/2009/06/090620_as_iraq_explosion_tc2			arabic	middleeast	2009	06	090620_as_iraq_explosion_tc2	nan	nan	090620_as_iraq_explosi
1	https://www.bbc.com/arabic/middleeast/2009/06/090620_iraq_blast_tc2	https	www.bbc.com	/arabic/middleeast/2009/06/090620_iraq_blast_tc2			arabic	middleeast	2009	06	090620_iraq_blast_tc2	nan	nan	090620_iraq_blast_tc2
2	https://www.bbc.com/arabic/business/2009/06/090622_me_worldbank_tc2	https	www.bbc.com	/arabic/business/2009/06/090622_me_worldbank_tc2			arabic	business	2009	06	090622_me_worldbank_tc2	nan	nan	090622_me_worldbank_
3	https://www.bbc.com/arabic/multimedia/2009/06/090624_me_inpictures_brazil_tc2	https	www.bbc.com	/arabic/multimedia/2009/06/090624_me_inpictures_brazil_tc2			arabic	multimedia	2009	06	090624_me_inpictures_brazil_tc2	nan	nan	090624_me_inpictures_
4	https://www.bbc.com/arabic/business/2009/06/090618_tomtest	https	www.bbc.com	/arabic/business/2009/06/090618_tomtest			arabic	business	2009	06	090618_tomtest	nan	nan	090618_tomtest
49994	https://www.bbc.com/vietnamese/world/2009/08/090831_dalailamataiwan	https	www.bbc.com	/vietnamese/world/2009/08/090831_dalailamataiwan			vietnamese	world	2009	08	090831_dalailamataiwan	nan	nan	090831_dalailamataiwan
49995	https://www.	https	www.bbc.com	/vietnamese/world/2009/09/090901_putin_regret_pact			vietnamese	world	2009	09	090901_putin_regret_pact	nan	nan	090901_putin_regret_pa

It seems that the dir_1 is where they have the language information, so we can easily count how many articles they have per language:

```
url_df['dir_1'].value_counts()
```

```
russian       14022
persian       10968
portuguese     5403
urdu           5068
mundo          5065
vietnamese     3561
arabic         2984
hindi          1677
turkce          706
ukchina         545
Name: dir_1, dtype: int64
```

We can also get a subset of articles written in a certain language, and see how many articles they publish per month, week, year, etc.

```
(bbc_sitemap[bbc_sitemap['loc']
 .str.contains('/russian/')]
 .set_index('lastmod')
 .resample('M')['loc'].count())
```

```
lastmod
2009-04-30 00:00:00+00:00    1506
2009-05-31 00:00:00+00:00    2910
2009-06-30 00:00:00+00:00    3021
2009-07-31 00:00:00+00:00    3250
2009-08-31 00:00:00+00:00    2769
                             ...
2019-09-30 00:00:00+00:00       8
2019-10-31 00:00:00+00:00      17
2019-11-30 00:00:00+00:00      11
2019-12-31 00:00:00+00:00      24
2020-01-31 00:00:00+00:00       6
Freq: M, Name: loc, Length: 130, dtype: int64
```

The topic or category of the article seems to be in dir_2 for which we can do the same and count the values.

```
url_df['dir_2'].value_counts()[:20]
```

```
rolling_news        9044
world               5050
noticias            4224
iran                3682
pakistan            2103
afghanistan         1959
multimedia          1657
internacional       1555
sport               1350
international       1293
```

(continues on next page)

(continued from previous page)

```
india               1285
america_latina      1274
business            1204
cultura_sociedad     913
middleeast           874
worldnews            872
russia               841
radio                769
science              755
football             674
Name: dir_2, dtype: int64
```

There is much more you can do, and a lot depends on the URL structure, which you have to explore and run the right operation.

For example, we can use the `last_dir` column which contains the slugs of the articles, replace underscores with spaces, split, concatenate all, put in a `pd.Series` and count the values. This way we see how many times each word occurred in an article. The same code can also be run after filtering for articles in a particular language to get a more meaningful list of words.

```
url_df['last_dir'].str.split('_').str[1:].explode().value_counts()[:20]
```

```
rn          8808
tc2         3153
iran        1534
video        973
obama        882
us           862
china        815
ir88         727
russia       683
si           640
np           638
afghan       632
ka           565
an           556
iraq         554
pakistan     547
nh           533
cq           520
zs           510
ra           491
Name: last_dir, dtype: int64
```

This was a quick overview and data preparation for a sample sitemap. Once you are familiar with the sitemap's structure, you can more easily start analyzing the content.

Note: There is a bug currently with tags that contain multiple values in sitemaps. If an image column in a news sitemap contains multiple images, only the last one is retreived. The same applies for any other sitemap that has a tag with multiple values.

1.7.3 News Sitemaps

```
nyt_news = adv.sitemap_to_df('https://www.nytimes.com/sitemaps/new/news.xml.gz')
print(nyt_news.shape)
# (5085, 16)
nyt_news
```

	loc	lastmod	news	news_publication_name	news_publication_language	news_publication_date	news_title	news_keywords	image	image_loc	sitemap	etag	sitemap_last_modified	sitemap_download_date
0	https://www.nytimes.com/interactive/2021/us/ottawa-ohio-covid-cases.html	2022-02-12 00:00:00+00:00		The New York Times	en	2021-01-27T17:00:00Z	Ottawa County, Ohio Covid Case and Exposure Risk Tracker	Coronavirus (2019-nCoV), States (US), Deaths (Fatalities), United States, Disease Rates	https://static01.nyt.com/images/sitemaps/2020/03/29/us/ohio-coronavirus-cases-promo-1585539358901.png	https://www.nytimes.com/sitemaps/new/news-6.xml.gz	0cff64520274c0179456	2022-02-12 20:17:31+00:00	2022-02-12 20:18:39.744247+00:00	78a888967d
1	https://www.nytimes.com/interactive/2021/us/hopewell-virginia-covid-cases.html	2022-02-12 00:00:00+00:00		The New York Times	en	2021-01-27T17:00:00Z	Hopewell, Virginia Covid Case and Exposure Risk Tracker	Coronavirus (2019-nCoV), States (US), Deaths (Fatalities), United States, Disease Rates	https://static01.nyt.com/images/sitemaps/2020/03/29/us/virginia-coronavirus-cases-promo-1585539536519.png	https://www.nytimes.com/sitemaps/new/news-6.xml.gz	0cff64520274c0179456	2022-02-12 20:17:31+00:00	2022-02-12 20:18:39.744247+00:00	78a888967d
2	https://www.nytimes.com/interactive/2021/us/box-butte-nebraska-covid-cases.html	2022-02-12 00:00:00+00:00		The New York Times	en	2021-01-27T17:00:00Z	Box Butte County, Nebraska Covid Case and Exposure Risk Tracker	Coronavirus (2019-nCoV), States (US), Deaths (Fatalities), United States, Disease Rates	https://static01.nyt.com/images/sitemaps/2020/03/29/us/nebraska-coronavirus-cases-promo-1585539237156.png	https://www.nytimes.com/sitemaps/new/news-6.xml.gz	0cff64520274c0179456	2022-02-12 20:17:31+00:00	2022-02-12 20:18:39.744247+00:00	78a888967d
3	https://www.nytimes.com/interactive/2021/us/stearns-minnesota-covid-cases.html	2022-02-12 00:00:00+00:00		The New York Times	en	2021-01-27T17:00:00Z	Stearns County, Minnesota Covid Case and Exposure Risk Tracker	Coronavirus (2019-nCoV), States (US), Deaths (Fatalities), United States, Disease Rates	https://static01.nyt.com/images/sitemaps/2020/03/29/us/minnesota-coronavirus-cases-promo-1585539237185.png	https://www.nytimes.com/sitemaps/new/news-6.xml.gz	0cff64520274c0179456	2022-02-12 20:17:31+00:00	2022-02-12 20:18:39.744247+00:00	78a888967d

1.7.4 Video Sitemaps

```
wired_video = adv.sitemap_to_df('https://www.wired.com/video/sitemap.xml')
print(wired_video.shape)
# (2955, 14)
wired_video
```

	loc	video	video_thumbnail_loc	video_title	video_description	video_content_loc	video_duration	video_publication_date	video_expiration_date	lastmod	sitemap	etag	sitemap_size_mb	download_date
0	https://www.wired.com/video/watch/autocomplete-inverviews-owen-wilson-answers-the-webs-most-searched-questions		http://dwgyu36up6iuz.cloudfront.net/heru80fdn/image/upload/c_fill,d_placeholder_thescene.jpg,fl_progressive,g_face,h_180,q_80,w_320/v1644595402/wired_autocomplete-inverviews-owen-wilson-answers-the-webs-most-searched-questions.jpg	Autocomplete Interview – Owen Wilson Answers The Web's Most Searched Questions	Owen Wilson takes the WIRED Autocomplete Interview and answers the internet's most searched questions about himself. How did Owen Wilson break his nose? How many movies is he in with Ben Stiller? Is Owen in every Wes Anderson movie? Is he a good skateboarder? Owen answers all these questions and much more!	http://dp8hsntg6do36.cloudfront.net/62067f085577c277dd9acf42/39687acb-505b-4c69-94f1-afaa7cb5e636low.mp4	645	2022-02-11T17:00:00+00:00	nan	NaT	https://www.wired.com/video/sitemap.xml	W/90b2…a2a57cb180cbd3c6f0	1.841798	2022-02-12 20:24:55.841851+00:00
1	https://www.wired.com/video/watch/wired-news-and-science-samsung-s22		http://dwgyu36up6iuz.cloudfront.net/heru80fdn/image/upload/c_fill,d_placeholder_thescene.jpg,fl_progressive,g_face,h_180,q_80,w_320/v1644418652/wired_wired-news-and-science-samsung-s22.jpg	Currents - Samsung S22 Ultra Explained in 3 Minutes	Julian Chokkattu, Reviews Editor for WIRED, walks us through a few of the Samsung S22 Ultra's new features.	http://dp8hsntg6do36.cloudfront.net/6203cd7b5577c23d19622259/fe546b9b-a320-4883-9cbd-0d790f23e36dlow.mp4	184	2022-02-10T17:00:00+00:00	nan	NaT	https://www.wired.com/video/sitemap.xml	W/90b2…a2a57cb180cbd3c6f0	1.841798	2022-02-12 20:24:55.841851+00:00
2	https://www.wired.com/video/watch/first-look-samsung-galaxy-unpacked-2022		http://dwgyu36up6iuz.cloudfront.net/heru80fdn/image/upload/	First Look: Samsung Galaxy Unpacked 2022	Samsung has debuted three new smartphones—the Galaxy S22 Ultra, S22+, S22—and three Android tablets	http://dp8hsntg6do36.cloudfront.net/620345a15577c23d46622256/d74930cf-11e1-466e-b023-1d9b9[illegible]1664204low.mp4	373	2022-02-09T15:00:00+00:00	nan	NaT	https://www.wired.com/video/sitemap.xml	W/90b2…a2a57cb180cbd3c6f0	1.841798	2022-02-12 20:24:55.841851+00:00

sitemap_to_df(*sitemap_url*, *max_workers=8*, *recursive=True*)

Retrieve all URLs and other available tags of a sitemap(s) and put them in a DataFrame.

You can also pass the URL of a sitemap index, or a link to a robots.txt file.

Parameters

- **sitemap_url** (*url*) -- The URL of a sitemap, either a regular sitemap, a sitemap index, or a link to a robots.txt file. In the case of a sitemap index or robots.txt, the function will go through all the sub sitemaps and retrieve all the included URLs in one DataFrame.
- **max_workers** (*int*) -- The maximum number of workers to use for threading. The higher the faster, but with high numbers you risk being blocked and/or missing some data as you might appear like an attacker.
- **recursive** (*bool*) -- Whether or not to follow and import all sub-sitemaps (in case you have a sitemap index), or to only import the given sitemap. This might be useful in case you want to explore what sitemaps are available after which you can decide which ones you are interested in.

Return sitemap_df A pandas DataFrame containing all URLs, as well as other tags if available (`lastmod`, `changefreq`, `priority`, or others found in news, video, or image sitemaps).

1.8 Python SEO Crawler / Spider

A customizable crawler to analyze SEO and content of pages and websites.

This is provided by the *crawl()* function which is customized for SEO and content analysis usage, and is highly configurable. The crawler uses Scrapy so you get all the power that it provides in terms of performance, speed, as well as flexibility and customization.

There are two main approaches to crawl:

1. **Discovery:** You know the website to crawl, so you provide a `url_list` (one or more URLs), and you want the crawler to go through the whole website(s) by following all available links.
2. **Pre-determined a.k.a "list mode":** You have a known set of URLs that you want to crawl and analyze, without following links or discovering new URLs.

1.8.1 Discovery Crawling Approach

The simplest way to use the function is to provide a list of one or more URLs and the crawler will go through all of the reachable pages.

```
>>> import advertools as adv
>>> adv.crawl('https://example.com', 'my_output_file.jl', follow_links=True)
```

That's it! To open the file:

```
>>> import pandas as pd
>>> crawl_df = pd.read_json('my_output_file.jl', lines=True)
```

What this does:

- Check the site's robots.txt file and get the crawl rules, which means that your crawl will be affected by these rules and the user agent you are using. Check the details below on how to change settings and user agents to control this.

- Starting with the provided URL(s) go through all links and parse pages.
- For each URL extract the most important SEO elements.
- Save them to `my_output_file.jl`.
- The column headers of the output file (once you import it as a DataFrame) would be the names of the elements (title, h1, h2, etc.).

Jsonlines is the supported output format because of its flexibility in allowing different values for different scraped pages, and appending indepentent items to the output files.

Note: When the crawler parses pages it saves the data to the specified file by appending, and not overwriting. Otherwise it would have to store all the data in memory, which might crash your computer. A good practice is to have a separate `output_file` for every crawl with a descriptive name *sitename_crawl_YYYY_MM_DD.jl* for example. If you use the same file you will probably get duplicate data in the same file.

1.8.2 Extracted On-Page SEO Elements

The names of these elements become the headers (column names) of the `output_file`.

Element	Remarks
url	The URL requested
title	The <title> tag(s)
viewport	The *viewport* meta tag if available
charset	The *charset* meta tag if available
meta_desc	Meta description
canonical	The canonical tag if available
alt_href	The *href* attribute of rel=alternate tags
alt_hreflang	The language codes of the alternate links
og:*	Open Graph data
twitter:*	Twitter card data
jsonld_*	JSON-LD data if available. In case multiple snippets occur, the respective column names will include a number
h1...h6	<*h1*> through <*h6*> tag(s), whichever is available
links_url	The URLs of the links on the page
links_text	The link text (anchor text)
links_nofollow	Boolean, whether or not the link is a nofllow link. Note that this only tells if the link itself contains a rel="nofoll
nav_links_text	The anchor text of all links in the <*nav*> tag if available
nav_links_url	The links in the <*nav*> tag if available
header_links_text	The anchor text of all links in the <*header*> tag if available
header_links_url	The links in the <*header*> tag if available
footer_links_text	The anchor text of all links in the <*footer*> tag if available
footer_links_url	The links in the <*footer*> tag if available
body_text	The text in the <p>, , and tags within <body>
size	The page size in bytes
download_latency	The amount of time it took to get the page HTML, in seconds.
download_timout	The amount of time (in secs) that the downloader will wait before timing out. Defaults to 180.
redirect_times	The number of times the pages was redirected if available
redirect_ttl	The default maximum number of redirects the crawler allows
redirect_urls	The chain of URLs from the requested URL to the one actually fetched
redirect_reasons	The type of redirection(s) 301, 302, etc.

Element	Remarks
depth	The depth of the current URL, relative to the first URLs where crawling started. The first pages to be crawled ha
status	Response status (200, 404, etc.)
img_*	All available <img> tag attributes. 'alt', 'crossorigin', 'height', 'ismap', 'loading', 'longdesc', 'referrerpolicy', 'sizes',
ip_address	IP address
crawl_time	Date and time the page was crawled
resp_headers_*	All available response headers (last modified, server, etc.)
request_headers_*	All available request headers (user-agent, encoding, etc.)

Note: All elements that may appear multiple times on a page (like heading tags, or images, for example), will be joined with two "@" signs @@. For example, **"first H2 tag@@second H2 tag@@third tag"** and so on. Once you open the file, you simply have to split by @@ to get the elements as a list.

Here is a sample file of a crawl of this site (output truncated for readability):

```
>>> import pandas as pd
>>> site_crawl = pd.read_json('path/to/file.jl', lines=True)
>>> site_crawl.head()
                              url                              title                   ␣
↪meta_desc                              h1                           h2                ␣
↪                h3                        body_text  size  download_timeout           ␣
↪  download_slot  download_latency  redirect_times  redirect_ttl                  ␣
↪redirect_urls redirect_reasons  depth  status                     links_href          ␣
↪          links_text                         img_src                          img_alt␣
↪   ip_address          crawl_time             resp_headers_date resp_headers_content-
↪type     resp_headers_last-modified resp_headers_vary    resp_headers_x-ms-request-id␣
↪resp_headers_x-ms-version resp_headers_x-ms-lease-status resp_headers_x-ms-blob-type␣
↪resp_headers_access-control-allow-origin   resp_headers_x-served resp_headers_x-
↪backend resp_headers_x-rtd-project resp_headers_x-rtd-version         resp_headers_x-
↪rtd-path  resp_headers_x-rtd-domain resp_headers_x-rtd-version-method resp_headers_x-
↪rtd-project-method resp_headers_strict-transport-security resp_headers_cf-cache-status␣
↪ resp_headers_age           resp_headers_expires resp_headers_cache-control          ␣
↪resp_headers_expect-ct resp_headers_server    resp_headers_cf-ray       resp_headers_cf-
↪request-id          request_headers_accept request_headers_accept-language       ␣
↪request_headers_user-agent request_headers_accept-encoding          request_headers_
↪cookie
0   https://advertools.readthedocs           advertools -  Python  Get productive as an␣
↪online ma  advertools@@Indices and tables  Online marketing productivity             ␣
↪                NaN   Generate keywords for SEM camp    NaN                NaN ␣
↪advertools.readthedocs.io               NaN              NaN            NaN  https://
↪advertools.readthedocs            [302]    NaN     NaN  #@@readme.html@@advertools.kw_␣
↪ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@               NaN                              ␣
↪     NaN  104.17.32.82  2020-05-21 10:39:35  Thu, 21 May 2020 10:39:35 GMT          ␣
↪    text/html  Wed, 20 May 2020 12:26:23 GMT   Accept-Encoding  720a8581-501e-0043-
↪01a2-2e77d2                2009-09-19                     unlocked                   ␣
↪BlockBlob                                         *  Nginx-Proxito-Sendfile          ␣
↪ web00007c                advertools                     master  /proxito/media/html/
↪advertools  advertools.readthedocs.io                        path                    ␣
↪          subdomain         max-age=31536000; includeSubDo                          HIT␣
↪              NaN  Thu, 21 May 2020 11:39:35 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596daca7dbaa7e9e-BUD ␣
↪02d86a3cea00007e9edb0cf2000000  text/html,application/xhtml+xm                      ␣
↪     en  Mozilla/5.0 (Windows NT 10.0;                     gzip, deflate  __
↪cfduid=d76b68d148ddec1efd004
```

(continues on next page)

(continued from previous page)

```
1   https://advertools.readthedocs            advertools -  Python                          ␣
↪      NaN                      advertools         Change Log - advertools  0.9.1 (2020-
↪05-19)@@0.9.0 (202   Ability to specify robots.txt    NaN               NaN ␣
↪advertools.readthedocs.io               NaN             NaN           NaN              ␣
↪               NaN              NaN    NaN     NaN  index.html@@readme.html@@adver ␣
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                                NaN           ␣
↪    NaN  104.17.32.82  2020-05-21 10:39:36  Thu, 21 May 2020 10:39:35 GMT            ␣
↪   text/html  Wed, 20 May 2020 12:26:23 GMT   Accept-Encoding  4f7bea3b-701e-0039-3f44-
↪2f1d9f                2009-09-19                        unlocked                      ␣
↪BlockBlob                                        *  Nginx-Proxito-Sendfile           ␣
↪ web00007h                advertools                      master  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                path              ␣
↪         subdomain         max-age=31536000; includeSubDo                            HIT␣
↪              NaN  Thu, 21 May 2020 11:39:35 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596daca9bcab7e9e-BUD ␣
↪02d86a3e0e00007e9edb0d72000000  text/html,application/xhtml+xm                        ␣
↪     en  Mozilla/5.0 (Windows NT 10.0;                   gzip, deflate  __
↪cfduid=d76b68d148ddec1efd004
2   https://advertools.readthedocs            advertools -  Python  Get productive as an␣
↪online ma  advertools@@Indices and tables  Online marketing productivity             ␣
↪               NaN   Generate keywords for SEM camp   NaN               NaN ␣
↪advertools.readthedocs.io               NaN             NaN           NaN              ␣
↪               NaN              NaN    NaN     NaN  #@@readme.html@@advertools.kw_ ␣
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                                NaN           ␣
↪    NaN  104.17.32.82  2020-05-21 10:39:36  Thu, 21 May 2020 10:39:35 GMT            ␣
↪   text/html  Wed, 20 May 2020 12:26:36 GMT   Accept-Encoding  98b729fa-e01e-00bf-24c3-
↪2e494d                2009-09-19                        unlocked                      ␣
↪BlockBlob                                        *  Nginx-Proxito-Sendfile           ␣
↪ web00007c                advertools                      latest  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                path              ␣
↪         subdomain         max-age=31536000; includeSubDo                            HIT␣
↪              NaN  Thu, 21 May 2020 11:39:35 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596daca9bf26d423-BUD ␣
↪02d86a3e150000d423322742000000  text/html,application/xhtml+xm                        ␣
↪     en  Mozilla/5.0 (Windows NT 10.0;                   gzip, deflate  __
↪cfduid=d76b68d148ddec1efd004
3   https://advertools.readthedocs    advertools package -  Python                      ␣
↪      NaN              advertools package     Submodules@@Module contents           ␣
↪               NaN   Top-level package for advertoo   NaN               NaN ␣
↪advertools.readthedocs.io               NaN             NaN           NaN              ␣
↪               NaN              NaN    NaN     NaN  index.html@@readme.html@@adver ␣
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                                NaN           ␣
↪    NaN  104.17.32.82  2020-05-21 10:39:36  Thu, 21 May 2020 10:39:35 GMT            ␣
↪   text/html  Wed, 20 May 2020 12:26:25 GMT   Accept-Encoding  7a28ef3b-801e-00c2-24c3-
↪2ed585                2009-09-19                        unlocked                      ␣
↪BlockBlob                                        *  Nginx-Proxito-Sendfile           ␣
↪ web000079                advertools                      master  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                path              ␣
↪         subdomain         max-age=31536000; includeSubDo                            HIT␣
↪              NaN  Thu, 21 May 2020 11:39:35 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596daca9bddb7ec2-BUD ␣
↪02d86a3e1300007ec2a808a2000000  text/html,application/xhtml+xm                        ␣
↪     en  Mozilla/5.0 (Windows NT 10.0;                   gzip, deflate
↪cfduid=d76b68d148ddec1efd004
```

(continues on next page)

(continued from previous page)

```
4   https://advertools.readthedocs   Python Module Index -  Python
↪      NaN             Python Module Index                              NaN
↪               NaN           © Copyright 2020, Eli   NaN               NaN
↪advertools.readthedocs.io               NaN             NaN           NaN
↪               NaN              NaN    NaN     NaN  index.html@@readme.html@@adver
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@               _static/minus.png
↪      -  104.17.32.82  2020-05-21 10:39:36  Thu, 21 May 2020 10:39:35 GMT
↪   text/html  Wed, 20 May 2020 12:26:23 GMT   Accept-Encoding  75911c9e-201e-00e6-34c3-
↪2e4ccb                2009-09-19                        unlocked
↪BlockBlob                                        *  Nginx-Proxito-Sendfile
↪ web00007g                 advertools                      master  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                path
↪         subdomain         max-age=31536000; includeSubDo                          HIT
↪              NaN  Thu, 21 May 2020 11:39:35 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596daca9b91fd437-BUD
↪02d86a3e140000d437b81532000000  text/html,application/xhtml+xm
↪     en  Mozilla/5.0 (Windows NT 10.0;                    gzip, deflate  __
↪cfduid=d76b68d148ddec1efd004
66  https://advertools.readthedocs  advertools.url_builders -  Pyt
↪      NaN  Source code for advertools.url                              NaN
↪               NaN           © Copyright 2020, Eli   NaN               NaN
↪advertools.readthedocs.io               NaN             NaN           NaN
↪               NaN              NaN    NaN     NaN  ../../index.html@@../../readme
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                              NaN
↪    NaN  104.17.32.82  2020-05-21 10:39:39  Thu, 21 May 2020 10:39:38 GMT
↪   text/html  Wed, 20 May 2020 12:26:36 GMT   Accept-Encoding  d99f2368-c01e-006f-18c3-
↪2ef5ef                2009-09-19                        unlocked
↪BlockBlob                                        *  Nginx-Proxito-Sendfile
↪ web00007a                 advertools                      latest  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                path
↪         subdomain         max-age=31536000; includeSubDo                          HIT
↪              NaN  Thu, 21 May 2020 11:39:38 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596dacbbb8afd437-BUD
↪02d86a494f0000d437b828b2000000  text/html,application/xhtml+xm
↪     en  Mozilla/5.0 (Windows NT 10.0;                    gzip, deflate  __
↪cfduid=d76b68d148ddec1efd004
67  https://advertools.readthedocs  advertools.kw_generate -  Pyth
↪      NaN  Source code for advertools.kw_                              NaN
↪               NaN           © Copyright 2020, Eli   NaN               NaN
↪advertools.readthedocs.io               NaN             NaN           NaN
↪               NaN              NaN    NaN     NaN  ../../index.html@@../../readme
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                              NaN
↪    NaN  104.17.32.82  2020-05-21 10:39:39  Thu, 21 May 2020 10:39:39 GMT
↪   text/html  Wed, 20 May 2020 12:26:36 GMT   Accept-Encoding  85855c48-c01e-00ce-13c3-
↪2e3b74                2009-09-19                        unlocked
↪BlockBlob                                        *  Nginx-Proxito-Sendfile
↪ web00007g                 advertools                      latest  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                path
↪         subdomain         max-age=31536000; includeSubDo                          HIT
↪              NaN  Thu, 21 May 2020 11:39:39 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596dacbd980bd423-BUD
↪02d86a4a7f0000d423323b42000000  text/html,application/xhtml+xm
↪     en  Mozilla/5.0 (Windows NT 10.0;                    gzip, deflate
↪cfduid=d76b68d148ddec1efd004
```

(continues on next page)

(continued from previous page)

```
68  https://advertools.readthedocs  advertools.ad_from_string -  P                    ␣
↪      NaN  Source code for advertools.ad_                              NaN          ␣
↪               NaN           © Copyright 2020, Eli   NaN              NaN ␣
↪advertools.readthedocs.io              NaN             NaN           NaN          ␣
↪               NaN             NaN    NaN     NaN  ../../index.html@@../../readme ␣
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                                NaN          ␣
↪    NaN  104.17.32.82  2020-05-21 10:39:39  Thu, 21 May 2020 10:39:39 GMT          ␣
↪   text/html  Wed, 20 May 2020 12:26:36 GMT   Accept-Encoding  b0aef497-801e-004a-1647-
↪2f6d5c                2009-09-19                        unlocked                  ␣
↪BlockBlob                                        *  Nginx-Proxito-Sendfile        ␣
↪ web00007k                 advertools                        latest  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                 path         ␣
↪         subdomain         max-age=31536000; includeSubDo                        HIT␣
↪              NaN  Thu, 21 May 2020 11:39:39 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596dacbd980cd423-BUD ␣
↪02d86a4a7f0000d423209db2000000  text/html,application/xhtml+xm                    ␣
↪     en  Mozilla/5.0 (Windows NT 10.0;                    gzip, deflate  __
↪cfduid=d76b68d148ddec1efd004
69  https://advertools.readthedocs  advertools.ad_create -  Python                  ␣
↪      NaN  Source code for advertools.ad_                              NaN          ␣
↪               NaN           © Copyright 2020, Eli   NaN              NaN ␣
↪advertools.readthedocs.io              NaN             NaN           NaN          ␣
↪               NaN             NaN    NaN     NaN  ../../index.html@@../../readme ␣
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                                NaN          ␣
↪    NaN  104.17.32.82  2020-05-21 10:39:39  Thu, 21 May 2020 10:39:39 GMT          ␣
↪   text/html  Wed, 20 May 2020 12:26:36 GMT   Accept-Encoding  9dfdd38a-101e-00a1-7ec3-
↪2e93a0                2009-09-19                        unlocked                  ␣
↪BlockBlob                                        *  Nginx-Proxito-Sendfile        ␣
↪ web00007c                 advertools                        latest  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                 path         ␣
↪         subdomain         max-age=31536000; includeSubDo                        HIT␣
↪              NaN  Thu, 21 May 2020 11:39:39 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596dacbd99847ec2-BUD ␣
↪02d86a4a7f00007ec2a811f2000000  text/html,application/xhtml+xm                    ␣
↪     en  Mozilla/5.0 (Windows NT 10.0;                    gzip, deflate  __
↪cfduid=d76b68d148ddec1efd004
70  https://advertools.readthedocs      advertools.emoji -  Python                  ␣
↪      NaN  Source code for advertools.emo                              NaN          ␣
↪               NaN           © Copyright 2020, Eli   NaN              NaN ␣
↪advertools.readthedocs.io              NaN             NaN           NaN          ␣
↪               NaN             NaN    NaN     NaN  ../../index.html@@../../readme ␣
↪@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                                NaN          ␣
↪    NaN  104.17.32.82  2020-05-21 10:39:40  Thu, 21 May 2020 10:39:39 GMT          ␣
↪   text/html  Wed, 20 May 2020 12:26:36 GMT   Accept-Encoding  2ad504a1-101e-000b-03c3-
↪2e454f                2009-09-19                        unlocked                  ␣
↪BlockBlob                                        *  Nginx-Proxito-Sendfile        ␣
↪ web000079                 advertools                        latest  /proxito/media/html/
↪advertools  advertools.readthedocs.io                                 path         ␣
↪         subdomain         max-age=31536000; includeSubDo                        HIT␣
↪              NaN  Thu, 21 May 2020 11:39:39 GMT       public, max-age=3600  max-
↪age=604800, report-uri="ht          cloudflare  596dacbd9fb97e9e-BUD ␣
↪02d86a4a7f00007e9edb13a2000000  text/html,application/xhtml+xm                    ␣
↪     en  Mozilla/5.0 (Windows NT 10.0;                    gzip, deflate
↪cfduid=d76b68d148ddec1efd004
```

(continues on next page)

(continued from previous page)

1.8.3 Pre-Determined Crawling Approach (List Mode)

Sometimes you might have a fixed set of URLs for which you want to scrape and analyze SEO or content performance. Some ideas:

SERP Data

Let's say you just ran *serp_goog* and got a bunch of top-ranking pages that you would like to analyze, and see how that relates to their SERP ranking.

You simply provide the `url_list` parameter and again specify the `output_file`. This will only crawl the specified URLs, and will not follow any links.

Now you have the SERP DataFrame, as well as the crawl output file. All you have to do is to merge them by the URL columns, and end up with a richer dataset

News Articles

You want to follow the latest news of a certain publication, and you extract their latest news URLs from their news sitemap using *sitemap_to_df* . You provide those URLs and crawl them only.

Google Analytics / Google Search Console

Since they provide reports for URLs, you can also combine them with the ones crawled and end up with a better perspective. You might be interested in knowing more about high bounce-rate pages, pages that convert well, pages that get less traffic than you think they should and so on. You can simply export those URLs and crawl them.

Any tool that has data about a set of URLs can be used.

Again running the function is as simple as providing a list of URLs, as well as a filepath where you want the result saved.

```
>>> adv.crawl(url_list, 'output_file.jl', follow_links=False)
```

The difference between the two approaches, is the simple parameter `follow_links`. If you keep it as `False` (the default), the crawler will only go through the provided URLs. Otherwise, it will discover pages by following links on pages that it crawls. So how do you make sure that the crawler doesn't try to crawl the whole web when `follow_links` is *True*? The `allowed_domains` parameter gives you the ability to control this, although it is an optional parameter. If you don't specify it, then it will default to only the domains in the `url_list` and their sub-domains if any. It's important to note that you have to set this parameter if you want to only crawl certain sub-domains.

1.8.4 Custom Extraction with CSS and XPath Selectors

The above approaches are generic, and are useful for exploratory SEO audits and the output is helpful for most cases.

But what if you want to extract special elements that are not included in the default output? This is extremely important, as there are key elements on pages that you need to additionally extract and analyze. Some examples might be tags, prices, social media shares, product price or availability, comments, and pretty much any element on a page that might be of interest to you.

For this you can use two special parameters for CSS and/or XPath selectors. You simply provide a dictionary *{'name_1': 'selector_1', 'name_2': 'selector_2'}* where the keys become the column names, and the values (selectors) will be used to extract the required elements.

I mostly rely on SlectorGadget which is a really great tool for getting the CSS/XPath selecotrs of required elements. In some pages it can get really tricky to figure that out. Other resources for learning more about selectors:

- Scrapy's documentaion for selectors
- CSS Selector Reference on W3C
- XPath tutorial on W3C

Once you have determined the elements that you want to extract and figured out what their names are going to be, you simply pass them as arguments to `css_selectors` and/or `xpath_selectors` as dictionaries, as decribed above.

Let's say you want to extract the links in the sidebar of this page. By default you would get all the links from the page, but you want to put those in the sidebar in a separate column. It seems that the CSS selector for them is *.toctree-l1 .internal*, and the XPath equivalent is *//*[contains(concat(" ", @class, " "), concat(" ", "toctree-l1", " "))]//*[contains(concat(" ", @class, " "), concat(" ", "internal", " "))]*. Note that this selects the *element* (the whole link object), which is not typically what you might be interested in.

So with CSS you need to append *::text* or *::attr(href)* if you want the text of the links or the *href* attribute respectively. Similarly with XPath, you will need to append */text()* or */@href* to the selector to get the same.

```
>>> adv.crawl('https://advertools.readthedocs.io/en/master/advertools.spider.html',
...           'output_file.jl',
...           css_selectors={'sidebar_links': '.toctree-l1 .internal::text',
...                          'sidebar_links_url': '.toctree-l1 .internal::attr(href)'})
```

Or, instead of `css_selectors` you can add a similar dictionary for the `xpath_selectors` argument:

```
>>> adv.crawl('https://advertools.readthedocs.io/en/master/advertools.spider.html',
...           'output_file.jl',
...           xpath_selectors={'sidebar_links': '//*[contains(concat( " ", @class, " " ),
↪ concat( " ", "toctree-l1", " " ))]//*[contains(concat( " ", @class, " " ), concat( "
↪", "internal", " " ))]/text()',
...                            'sidebar_links_url': '//*[contains(concat( " ", @class, "
↪" ), concat( " ", "toctree-l1", " " ))]//*[contains(concat( " ", @class, " " ),␣
↪concat( " ", "internal", " " ))]/@href'})
```

1.8.5 Customizing the Crawling Behavior while Following Links

If you set `follow_links=True` you might want to restrict this option for a more customized crawl. This means you can decide whether to include or exclude certain links based on certain conditions based on URL parameters and/or URL regex patterns.

URL Query Parameters

Two options are available for this:

1. `exclude_url_params`: By providing a list of URL parameters, any link that contains any of the parameters will not be followed. For example if you set this option to ['price', 'country'], and the page currently being crawled contains three links:

- **/shoes/model_a?price=10&country=us**: Contains both parameters, will not be followed.
- **/shoes/model_a?price=10**: Contains one of the parameters, will not be followed.
- **/shoes/model_b?color=black**: Contains "color" which was not listed, will be followed

To make this efficient, and in case you want to skip all links that contain any parameter, you can set this option to `True`, and any link that contains any URL parameter will not be followed `exclude_url_params=True`

2. `include_url_params`: Similarly, you can choose the parameters that links should contain in order for them to be followed. If a link contains any of the listed parameters, that link will be followed. Even though this option is straightforward, it might give you unexpected results. If you set the parameters to include as ['price'] for example, and you start crawling from a page that doesn't have any link containing that parameter, crawling will stop. Yet, the website might have many links with that parameter. Please keep this in mind, and remember that reasoning about the exclude option is easier than the include option for this reason/example.

URL Regex Patterns

You might want even more granular control over which links to follow, and might be interested in other URL properties than their query parameters. You have two simple options to include/exclude links based on whether they match a certain regex pattern.

1. `exclude_url_regex`: Enter a regex, and the links will be checked if they match it. If they do, they will not be followed, if not they will.

2. `include_url_regex`: This is similar but tells the crawler which links to follow, based on whether or not they match the regex. This option also has the same potentially tricky behavior like the `include_url_params` option.

Here is a simple example showing how you might control following links using all four options:

```
import advertools as adv
adv.crawl('https://example.com', 'output_file.jl', follow_links=True,

          # don't follow links containing any of these parameters:
          exclude_url_params=['price', 'region'],
          # follow links containing any of these parameters:
          include_url_params=['source'],
          # don't follow links that contain the pattern "/fr/" or "/de/":
          exclude_url_regex='/fr/|/de/',
          # follow links that contain the pattern "/shop/":
          include_url_regex='/shop/'
          )
```

1.8.6 Spider Custom Settings and Additional Functionality

In addition to what you can control regarding the items you can extract, you can also customize the behaviour of the spider and set rules for crawling so you can control it even further.

This is provided by the `custom_settings` parameter. It is optional, and takes a dictionary of settings and their values. Scrapy provides a very large number of settings, and they are all available through this parameter (assuming some conditions for some of the settings).

Here are some examples that you might find interesting:

- *CONCURRENT_REQUESTS_PER_DOMAIN* Defaults to 8, and controls the number of simultaneous requests to be performed for each domain. You might want to lower this if you don't want to put too much pressure on the website's server, and you probably don't want to get blocked!
- *DEFAULT_REQUEST_HEADERS* You can change this if you need to.
- *DEPTH_LIMIT* How deep your crawl will be allowed. The default has no limit.
- *DOWNLOAD_DELAY* Similar to the first option. Controls the amount of time in seconds for the crawler to wait between consecutive pages of the same website. It can also take fractions of a second (0.4, 0.75, etc.)
- *LOG_FILE* If you want to save your crawl logs to a file, which is strongly recommended, you can provide a path to it here.
- *USER_AGENT* If you want to identify yourself differently while crawling. This is affected by the robots.txt rules, so you would be potentially allowed/disallowed from certain pages based on your user-agent.
- *CLOSESPIDER_ERRORCOUNT*, *CLOSESPIDER_ITEMCOUNT*, *CLOSESPIDER_PAGECOUNT*, *CLOSESPIDER_TIMEOUT* Stop crawling after that many errors, items, pages, or seconds. These can be very useful to limit your crawling in certain cases. I particularly like to use *CLOSESPIDER_PAGECOUNT* when exploring a new website, and also to make sure that my selectors are working as expected. So for your first few crawls you might set this to five hundred for example and explore the crawled pages. Then when you are confident things are working fine, you can remove this restriction. *CLOSESPIDER_ERRORCOUNT* can also be very useful while exploring, just in case you get unexpected errors.

The next page contains a number of *strategies and recipes for crawling* with code examples and explanations.

Usage

A very simple dictionary to be added to your function call:

```
>>> adv.crawl('http://exmaple.com', 'outpuf_file.jl',
...           custom_settings={'CLOSESPIDER_PAGECOUNT': 100,
...                            'CONCURRENT_REQUESTS_PER_DOMAIN': 1,
...                            'USER_AGENT': 'custom-user-agent'})
```

Please refer to the spider settings documentation for the full details.

crawl(*url_list*, *output_file*, *follow_links=False*, *allowed_domains=None*, *exclude_url_params=None*, *include_url_params=None*, *exclude_url_regex=None*, *include_url_regex=None*, *css_selectors=None*, *xpath_selectors=None*, *custom_settings=None*)

Crawl a website's URLs based on the given `url_list`

Parameters

- **url_list** (*url, list*) -- One or more URLs to crawl. If `follow_links` is True, the crawler will start with these URLs and follow all links on pages recursively.
- **output_file** (*str*) -- The path to the output of the crawl. Jsonlines only is supported to allow for dynamic values. Make sure your file ends with ".jl", e.g. *output_file.jl*.

- **follow_links** (*bool*) -- Defaults to False. Whether or not to follow links on crawled pages.
- **exclude_url_params** (*list, bool*) -- A list of URL parameters to exclude while following links. If a link contains any of those parameters, don't follow it. Setting it to `True` will exclude links containing any parameter.
- **include_url_params** (*list*) -- A list of URL parameters to include while following links. If a link contains any of those parameters, follow it. Having the same parmeters to include and exclude raises an error.
- **exclude_url_regex** (*str*) -- A regular expression of a URL pattern to exclude while following links. If a link matches the regex don't follow it.
- **include_url_regex** (*str*) -- A regular expression of a URL pattern to include while following links. If a link matches the regex follow it.
- **css_selectors** (*dict*) -- A dictionary mapping names to CSS selectors. The names will become column headers, and the selectors will be used to extract the required data/content.
- **xpath_selectors** (*dict*) -- A dictionary mapping names to XPath selectors. The names will become column headers, and the selectors will be used to extract the required data/content.
- **custom_settings** (*dict*) -- A dictionary of optional custom settings that you might want to add to the spider's functionality. There are over 170 settings for all kinds of options. For details please refer to the spider settings documentation.
- **allowed_domains** (*list*) -- (optional) A list of the allowed domains to crawl. This ensures that the crawler does not attempt to crawl the whole web. If not specified, it defaults to the domains of the URLs provided in `url_list` and all their sub-domains. You can also specify a list of sub-domains, if you want to only crawl those.

Examples

Crawl a website and let the crawler discover as many pages as available

```
>>> import advertools as adv
>>> adv.crawl('http://example.com', 'output_file.jl', follow_links=True)
>>> import pandas as pd
>>> crawl_df = pd.read_json('output_file.jl', lines=True)
```

Crawl a known set of pages (on a single or multiple sites) without following links (just crawl the specified pages) or "list mode":

```
>>> adv.crawl(['http://exmaple.com/product', 'http://exmaple.com/product2',
...            'https://anotherexample.com', 'https://anotherexmaple.com/hello'],
...            'output_file.jl', follow_links=False)
```

Crawl a website, and in addition to standard SEO elements, also get the required CSS selectors. Here we will get three additional columns *price*, *author*, and *author_url*. Note that you need to specify if you want the text attribute or the *href* attribute if you are working with links (and all other selectors).

```
>>> adv.crawl('http://example.com', 'output_file.jl',
...           css_selectors={'price': '.a-color-price::text',
...                          'author': '.contributorNameID::text',
...                          'author_url': '.contributorNameID::attr(href)'})
```

1.9 SEO Crawling & Scraping: Strategies & Recipes

Once you have mastered the basics of using the *crawl* function, you probably want to achieve more with better customization and control.

These are some code strategies that might be useful to customize how you run your crawls.

Most of these options can be set using the `custom_settings` parameter that the function takes. This can be set by using a dictionary, where the keys indicate the option you want to set, and the values specify how you want to set them.

1.9.1 How to crawl a list of pages, and those pages only (list mode)?

Simply provide that list as the first argument, for the `url_list` parameter, and make sure that `follow_links=False`, which is the default. This simply crawls the given pages, and stops when done.

```
>>> import advertools as adv
>>> url_list = ['https://example.com/page_1',
...             'https://example.com/page_2',
...             'https://example.com/page_3',
...             'https://example.com/page_4']

>>> adv.crawl(url_list,
...           output_file='example_crawl_1.jl',
...           follow_links=False)
```

1.9.2 How can I crawl a website including its sub-domains?

The *crawl* function takes an optional `allowed_domains` parameter. If not provided, it defaults to the domains of the URLs in `url_list`. When the crawler goes through the pages of *example.com*, it follows links to discover pages. If it finds pages on *help.exmaple.com* it won't crawl them (it's a different domain). The solution, therefore, is to provide a list of domains to the `allowed_domains` parameter. Make sure you also include the original domain, in this case *example.com*.

```
>>> adv.crawl('https://example.com',
...           'example_crawl_1.jl',
...           follow_links=True
...           allowed_domains=['help.example.com', 'example.com', 'community.example.com
↪'])
```

1.9.3 How can I save a copy of the logs of my crawl for auditing them later?

It's usually good to keep a copy of the logs of all your crawls to check for errors, exceptions, stats, etc. Pass a path of the file where you want the logs to be saved, in a dictionary to the `cutom_settings` parameter. A good practice for consistency is to give the same name to the `output_file` and log file (with a different extension) for easier retreival. For example:

`output_file`: 'website_name_crawl_1.jl'
`LOG_FILE`: 'website_name_crawl_1.log' (.txt can also work)
`output_file`: 'website_name_crawl_2.jl'
`LOG_FILE`: 'website_name_crawl_2.log'

```
>>> adv.crawl('https://example.com', 'example_crawl_1.jl',
...           custom_settings={'LOG_FILE': 'example_crawl_1.log'})
```

1.9.4 How can I automatically stop my crawl based on a certain condition?

There are a few conditions that you can use to trigger the crawl to stop, and they mostly have descriptive names:

- CLOSESPIDER_ERRORCOUNT: You don't want to wait three hours for a crawl to finish, only to discover that you had errors all over the place. Set a certain number of errors to trigger the crawler to stop, so you can investigate the issue.
- CLOSESPIDER_ITEMCOUNT: Anything scraped from a page is an "item", h1, title , meta_desc, etc. Set the crawler to stop after getting a certain number of items if you want that.
- CLOSESPIDER_PAGECOUNT: Stop the crawler after a certain number of pages have been crawled. This is useful as an exploratory technique, especially with very large websites. It might be good to crawl a few thousand pages, get an idea on its structure, and then run a full crawl with those insights in mind.
- CLOSESPIDER_TIMEOUT: Stop the crawler after a certain number of seconds.

```
>>> adv.crawl('https://example.com', 'example_crawl_1.jl',
...           custom_settings={'CLOSESPIDER_PAGECOUNT': 500})
```

1.9.5 How can I (dis)obey robots.txt rules?

The crawler obeys robots.txt rules by default. Sometimes you might want to check the results of crawls without doing that. You can set the ROBOTSTXT_OBEY setting under custom_settings:

```
>>> adv.crawl('https://example.com',
...           'example_crawl_1.jl',
...           custom_settings={'ROBOTSTXT_OBEY': False})
```

1.9.6 How do I set my User-agent while crawling?

Set this parameter under *custom_settings* dictionary under the key USER_AGENT. The default User-agent can be found by running *adv.spider.user_agent*

```
>>> adv.spider.user_agent # to get the current User-agent
>>> adv.crawl('http://example.com',
...           'example_crawl_1.jl',
...           custom_settings={'USER_AGENT': 'YOUR_USER_AGENT'})
```

1.9.7 How can I control the number of concurrent requests while crawling?

Some servers are set for high sensitivity to automated and/or concurrent requests, that you can quickly be blocked/banned. You also want to be polite and not kill those servers, don't you?

There are several ways to set that under the `custom_settings` parameter. The available keys are the following:

`CONCURRENT_ITEMS`: default 100
`CONCURRENT_REQUESTS` : default 16
`CONCURRENT_REQUESTS_PER_DOMAIN`: default 8
`CONCURRENT_REQUESTS_PER_IP`: default 0

```
>>> adv.crawl('https://example.com',
...           'example_crawl_1.jl',
...           custom_settings={'CONCURRENT_REQUESTS_PER_DOMAIN': 1})
```

1.9.8 How can I slow down the crawling so I don't hit the websites' servers too hard?

Use the `DOWNLOAD_DELAY` setting and set the interval to be waited before downloading consecutive pages from the same website (in seconds).

```
>>> adv.crawl('https://example.com', 'example_crawl_1.jl',
...           custom_settings={'DOWNLOAD_DELAY': 3}) # wait 3 seconds between pages
```

1.9.9 How can I set multiple settings to the same crawl job?

Simply add multiple settings to the `custom_settings` parameter.

```
>>> adv.crawl('http://example.com',
...           'example_crawl_1.jl',
...           custom_settings={'CLOSESPIDER_PAGECOUNT': 400,
...                            'CONCURRENT_ITEMS': 75,
...                            'LOG_FILE': 'output_file.log'})
```

1.9.10 I want to crawl a list of pages, follow links from those pages, but only to a certain specified depth

Set the `DEPTH_LIMIT` setting in the `custom_settings` parameter. A setting of 1 would follow links one level after the provided URLs in `url_list`

```
>>> adv.crawl('http://example.com',
...           'example_crawl_1.jl',
...           custom_settings={'DEPTH_LIMIT': 2}) # follow links two levels from the␣
↪initial URLs, then stop
```

1.9.11 How do I pause/resume crawling, while making sure I don't crawl the same page twice?

There are several reasons why you might want to do this:

- You want to mainly crawl the updates to the site (you already crawled the site).
- The site is very big, and can't be crawled quickly.
- You are not in a hurry, and you also don't want to hit the servers hard, so you run your crawl across days for example.
- As an emergency measure (connection lost, battery died, etc.) you can start where you left off

Handling this is extremely simple, and all you have to do is simply provide a path to a new folder. Make sure it is new and empty, and make sure to only use it for the same crawl job reruns. That's all you have to worry about. The `JOBDIR` setting handles this.

```
>>> adv.crawl('http://example.com',
...           'example_crawl_1.jl',
...           custom_settings={'JOBDIR': '/Path/to/en/empty/folder'})
```

The first time you run the above code and then stop it. Stopping can happen by accident (lost connection, closed computer, etc.), manually (you hit ctrl+C) or you used a custom setting option to stop the crawl after a certain number of pages, seconds, etc.

The second time you want to run this, you simply run the exact same command again. If you check the folder that was created you can see a few files that manage the process. You don't need to worry about any of it. But make sure that folder doesn't get changed manually, rerun the same command as many times as you need, and the crawler should handle de-duplication for you.

1.9.12 XPath expressions for custom extraction

The following are some expressions you might find useful in your crawling, whether you use `advertools` or not. The first column indicates whether or not the respective expression is used by default by the `advertools` crawler.

Used by advertools	Suggested Name	XPath Expression	What it does
True	title	//title/text()	Extract the text
True	meta_desc	//meta[@name='description']/@content	Extract the con
True	viewport	//meta[@name='viewport']/@content	Extract the con
True	charset	//meta[@charset]/@charset	Get the meta ta
True	h1	//h1/text()	Get the h1 tags
True	h2	//h2/text()	Get the h2 tags
True	h3	//h3/text()	Get the h3 tags
True	h4	//h4/text()	Get the h4 tags
True	h5	//h5/text()	Get the h5 tags
True	h6	//h6/text()	Get the h6 tags
True	canonical	//link[@rel='canonical']/@href	Get the link ele
True	alt_href	//link[@rel='alternate']/@href	Get the link ele
True	alt_hreflang	//link[@rel='alternate']/@hreflang	Get the link ele
True	og_props	//meta[starts-with(@property, 'og:')]/@property	Extract all prop
True	og_content	//meta[starts-with(@property, 'og:')]/@content	Extract the con
True	twtr_names	//meta[starts-with(@name, 'twitter:')]/@name	Get meta tags w
True	twtr_content	//meta[starts-with(@name, 'twitter:')]/@content	Get meta tags w

Table 2 – continued from previous pa

Used by advertools	Suggested Name	XPath Expression	What it does
False	iframe_src	//iframe/@src	Get the iframes
False	gtm_script	//script[contains(@src, 'googletagmanager.com/gtm.js?id=')]/@src	Get the script w
False	gtm_noscript	//iframe[contains(@src, 'googletagmanager.com/ns.html?id=')]/@src	Get the iframes
False	link_rel_rel	//link[@rel]/@rel	Get all the link
False	link_rel_href	//link[@rel]/@href	Get all the link
False	link_rel_stylesheet	//link[@rel='stylesheet']/@href	Get all the link
False	css_links	//link[contains(@href, '.css')]/@href	Get the link ele
True	nav_links_text	//nav//a/text()	From the nav el
True	nav_links_href	//nav//a/@href	From the nav el
True	header_links_text	//header//a/text()	From the heade
True	header_links_href	//header//a/@href	From the heade
True	footer_links_text	//footer//a/text()	From the footer
True	footer_links_href	//footer//a/@href	From the footer
False	js_script_src	//script[@type='text/javascript']/@src	From script tag
False	js_script_text	//script[@type='text/javascript']/text()	From script tag
False	script_src	//script//@src	Get the src attri
False	canonical_parent	name(//link[@rel='canonical']/..)	Get the name o

1.10 Python Status Code Checker with Response Headers

A mini crawler that only makes `HEAD` requests to a known list of URLs. It uses Scrapy under the hood, which means you get all its power in a simplified interface for a simple and specific use-case.

The *`crawl_headers()`* function can be used to make those requests for various quality assurance and analysis reasons. Since `HEAD` requests don't download the whole page, this makes the crawling super light on servers, and makes the process very fast.

The function is straight-forward and easy to use, you basically need a list of URLs and a file path where you want to save the output (in *.jl* format):

```
import advertools as adv
import pandas as pd

url_list = ['https://advertools.readthedocs.io', 'https://adver.tools',
            'https://www.dashboardom.com', 'https://povertydata.org']
adv.crawl_headers(url_list, 'output_file.jl')
headers_df = pd.read_json('output_file.jl', lines=True)

headers_df
```

	url							col						
0	https://advertools	2022-02-11 02:32:26	200	180	advertools	0.027	[illegible]	HTTP/1.1	nan	[illegible]	nginx/1.18.0 (Ubuntu)	Fri, 11 Feb 2022 02:32:26 GMT	text/html; charset=utf-8	[illegible]
1	https://povertydata.org	2022-02-11 02:32:26	200	180	povertydata.org	0.0644	[illegible]	HTTP/1.1	nan	[illegible]	nginx/1.18.0 (Ubuntu)	Fri, 11 Feb 2022 02:32:26 GMT	text/html; charset=utf-8	[illegible]
2	https://advertools.readthedocs.io/en/master/	2022-02-11 02:32:26	200	180	advertools.readthedocs.io	0.027	[illegible]	HTTP/1.1	nan	[illegible]	cloudflare	Fri, 11 Feb 2022 02:32:26 GMT	[illegible]	[illegible]
3	https://www.dashboardom.com	2022-02-11 02:32:26	200	180	www.dashboardom.com	0.08	[illegible]	HTTP/1.1	nan	[illegible]	gunicorn/19.9.0	Fri, 11 Feb 2022 02:32:26 GMT	text/html; charset=utf-8	[illegible]

Optionally, you can customize the crawling behavior with the optional `custom_settings` parameter. Please check the crawl strategies page for tips on how you can do that.

Here are some of the common reasons for using a `HEAD` crawler:

- **Checking status codes:** One of the most important maintenance tasks you should be doing continuously. It's very easy to set up an automated script the checks status codes for a few hundred or thousand URLs on a periodic basis. You can easily build some rules and alerts based on the status codes you get.
- **Status codes of page elements:** Yes, your page returns a 200 OK status, but what about all the elements/components of the page? Images, links (internal and external), hreflang, canonical, URLs in metatags, script URLs, URLs in various structured data elements like Twitter, OpenGraph, and JSON-LD are some of the most important ones to check as well.
- **Getting search engine directives:** Those directives can be set using meta tags as well as response headers. This crawler gets all available response headers so you can check for search engine-specific ones, like *noindex* for example.
- **Getting image sizes:** You might want to crawl a list of image URLs and get their meta data. The response header *Content-Length* contains the length of the page in bytes. With images, it contains the size of the image. This can be an extremely efficient way of analyzing image sizes (and other meta data) without having to download those

images, which could consume a lot of bandwidth. Lookout for the column `resp_headers_content-length`.

- **Getting image types:** The `resp_headers_content-type` gives you an indication on the type of content of the page (or image when crawling image URLs); *text/html*, *image/jpeg* and *image/png* are some such content types.

class HeadersSpider(**args*, ***kwargs*)

Bases: `scrapy.spiders.Spider`

custom_settings: Optional[dict] = {'HTTPERROR_ALLOW_ALL': True, 'ROBOTSTXT_OBEY': True, 'USER_AGENT': 'advertools/0.13.1'}

errback(*failure*)

name: Optional[str] = 'headers_spider'

parse(*response*)

start_requests()

crawl_headers(*url_list*, *output_file*, *custom_settings=None*)

Crawl a list of URLs using the HEAD method.

This function helps in analyzing a set of URLs by getting status codes, download latency, all response headers and a few other meta data about the crawled URLs.

Sine the full page is not downloaded, these requests are very light on servers and it is super-fast. You can modify the speed of course through various settings.

Typically status code checking is an on-going task that needs to be done and managed. Automated alerts can be easily created based on certain status codes. Another interesting piece of the information is the *Content-Length* response header. This gives you the size of the response body without having to download the whole page. It can also be very interesting with image URLs. Downloading all images can really be expensive and time consuming. Being able to get image sizes without having to download them can help a lot in making decisions about optimizing those images. Several other data can be interesting to analyze, depending on what response headers you get.

Parameters

- **url_list** (*url,list*) -- One or more URLs to crawl.
- **output_file** (*str*) -- The path to the output of the crawl. Jsonlines only is supported to allow for dynamic values. Make sure your file ends with ".jl", e.g. *output_file.jl*.
- **custom_settings** (*dict*) -- A dictionary of optional custom settings that you might want to add to the spider's functionality. There are over 170 settings for all kinds of options. For details please refer to the spider settings documentation.

Examples

```
>>> import advertools as adv
>>> url_list = ['https://exmaple.com/A', 'https://exmaple.com/B',
...             'https://exmaple.com/C', 'https://exmaple.com/D',
...             'https://exmaple.com/E']
```

```
>>> adv.crawl_headers(url_list, 'output_file.jl')
>>> import pandas as pd
>>> crawl_df = pd.read_json('output_file.jl', lines=True)
```

1.11 Log File Analysis

Logs contain very detailed information about events happening on computers. And the extra details that they provide come with additional complexity that we need to handle ourselves. A pageview may contain many log lines, and a session can consist of several pageviews for example.

Another important characterisitic of log files is that their are usualy not big. They are massive.

So, we also need to cater for their large size, as well as rapid changes.

TL;DR

```
>>> import advertools as adv
>>> import pandas as pd
>>> adv.logs_to_df(log_file='access.log',
...                output_file='access_logs.parquet',
...                errors_file='log_errors.csv',
...                log_format='common',
...                fields=None)
>>> logs_df = pd.read_parquet('access_logs.parquet')
```

1.11.1 How to run the `logs_to_df()` function:

- `log_file`: The path to the log file you are trying to analyze.
- `output_file`: The path to where you want the parsed and compressed file to be saved. Only the *parquet* format is supported.
- `errors_file`: You will almost certainly have log lines that don't conform to the format that you have, so all lines that weren't properly parsed would go to this file. This file also contains the error messages, so you know what went wrong, and how you might fix it. In some cases, you might simply take these "errors" and parse them again. They might not be really errors, but lines in a different format, or temporary debug messages.
- `log_format`: The format in which your logs were formatted. Logs can (and are) formatted in many ways, and there is no right or wrong way. However, there are defaults, and a few popular formats that most servers use. It is likely that your file is in one of the popular formats. This parameter can take any one of the pre-defined formats, for example "common", or "combined", or a regular expression that you provide. This means that **you can parse any log format** (as long as lines are single lines, and not formatted in JSON).
- `fields`: If you selected one of the supported formats, then there is no need to provide a value for this parameter. You have to provide a list of fields in case you provide a custom (regex) format. The fields will become the names of the columns of the resulting DataFrame, so you can distinguish between them (client, time, status code, response size, etc.)

1.11.2 Supported Log Formats

- *common*
- *combined* (a.k.a "extended")
- *common_with_vhost*
- *nginx_error*
- *apache_error*

1.11.3 Log File Analysis - Data Preparation

We go through an example where we prepare the data for analysis, and here is the plan:

1. Parse the log file into a DataFrame saved to disk with a *.parquet* extension. A side effect is that your log file is also compressed down to 5% - 15% of its original size. It also makes it super efficient to query and analyze once in this format. Function used: `logs_to_df`.
2. Convert data types as needed (optional): Most importantly converting the *datetime* column into a date object helps a lot in querying the data. Other possibilities include converting to categorical data types for more efficient storage and querying. Function used: `pandas.to_datetime`.
3. Get the hostnames of the IP addresses of the clients sending requests. Function used: *reverse_dns_lookup*. We can then easily add a `hostname` column to the original DataFrame.
4. Parse and split URL columns into their respective components. Typically we have `request` which is the resource/URL requested, as well as `referer` , which shows us where the request was referred from. Function used: *url_to_df* .
5. Parse user agents if available. This allows us to analyze by user-agent family, operating system, bot/non-bot, version, and any other combination we want.
6. Combine all data together, and save back to a new *.parquet* file, and start analyzing.

```
!head data/sample_log.log
```

```
66.249.73.72 - - [16/Feb/2022:00:18:53 +0000] "GET / HTTP/1.1" 200 1095 "-" "Mozilla/5.0␣
↪(Linux; Android 6.0.1; Nexus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/98.0.4758.80 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.
↪google.com/bot.html)"
109.237.103.118 - - [16/Feb/2022:00:20:39 +0000] "GET /.env HTTP/1.1" 404 209 "-"
↪"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.
↪4044.129 Safari/537.36"
45.12.223.214 - - [16/Feb/2022:00:23:45 +0000] "GET / HTTP/1.0" 200 2240 "http://adver.
↪tools/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/90.0.4430.72 Safari/537.36"
51.68.77.249 - - [16/Feb/2022:00:26:23 +0000] "GET /robots.txt HTTP/1.1" 404 209 "-"
↪"advertools/0.13.0"
51.68.77.249 - - [16/Feb/2022:00:26:23 +0000] "HEAD / HTTP/1.1" 200 0 "-" "advertools/0.
↪13.0"
192.241.211.176 - - [16/Feb/2022:00:31:16 +0000] "GET /login HTTP/1.1" 404 209 "-"
↪"Mozilla/5.0 zgrab/0.x"
66.249.73.69 - - [16/Feb/2022:00:48:56 +0000] "GET /robots.txt HTTP/1.1" 404 209 "-"
↪"Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
66.249.73.72 - - [16/Feb/2022:00:48:56 +0000] "GET /staging/urlytics/ HTTP/1.1" 200 520
↪"-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
66.249.73.75 - - [16/Feb/2022:00:49:38 +0000] "GET /staging/urlytics/_dash-component-
↪suites/dash/html/dash_html_components.v2_0_0m1638886228.min.js HTTP/1.1" 200 154258
↪"http://www.adver.tools/staging/urlytics/" "Mozilla/5.0 AppleWebKit/537.36 (KHTML,␣
↪like Gecko; compatible; Googlebot/2.1; +http://www.google.com/bot.html) Chrome/98.0.
↪4758.80 Safari/537.36"
66.249.73.75 - - [16/Feb/2022:00:49:39 +0000] "GET /staging/urlytics/_dash-layout HTTP/1.
↪1" 200 2547 "http://www.adver.tools/staging/urlytics/" "Mozilla/5.0 AppleWebKit/537.36␣
↪(KHTML, like Gecko; compatible; Googlebot/2.1; +http://www.google.com/bot.html) Chrome/
↪98.0.4758.80 Safari/537.36"
```

```
import advertools as adv
import pandas as pd
from ua_parser import user_agent_parser
pd.options.display.max_columns = None

adv.logs_to_df(log_file='data/sample_log.log',
               output_file='data/adv_logs.parquet',
               errors_file='data/adv_errors.txt',
               log_format='combined')
```

Read the parquet file into a pandas DataFrame, and convert the `datetime` column into a datetime object.

```
logs_df = pd.read_parquet('data/adv_logs.parquet')
logs_df['datetime'] = pd.to_datetime(logs_df['datetime'],
                                     format='%d/%b/%Y:%H:%M:%S %z')

logs_df
```

Perform a reverse DNS lookup on the IP addresses in the `client` column:

```
%%time
host_df = adv.reverse_dns_lookup(logs_df['client'])
print(f'Rows, columns: {host_df.shape}')
host_df.head(15)
# Rows, columns: (1210, 9)
# CPU times: user 745 ms, sys: 729 ms, total: 1.47 s
# Wall time: 21.1 s
```

	ip_address	count	cum_count	perc	cum_perc	hostname	aliaslist	ipad-drlist	errors
0	143.244.132.26	426	426	0.070100	4.070100				[Errno 1] Unknown host
1	45.146.164.210	290	716	0.047720	9.117821				[Errno 1] Unknown host
2	46.177.196.171	192	908	0.031594	5.149416	ppp046177196171.access.hol.gr	171.196.177.46.in-addr.arpa	46.177.196.171	
3	185.22.173.88	182	1090	0.029949	0.179365				[Errno 1] Unknown host
4	152.32.226.223	171	1261	0.028138	9.207504				[Errno 1] Unknown host
5	94.200.35.174	154	1415	0.025341	5.232845				[Errno 1] Unknown host
6	89.47.44.105	130	1545	0.021392	0.254237	ppp089047044105.access.hol.gr	105.44.47.89.in-addr.arpa	89.47.44.105	
7	94.200.92.211	119	1664	0.019582	0.273819				[Errno 1] Unknown host
8	143.244.132.234	113	1777	0.018594	7.292414				[Errno 1] Unknown host
9	217.100.98.101	81	1858	0.013328	9.305743	d9646265.static.ziggozakelijk.nl	101.98.100.217.in-addr.arpa	217.100.98.101	
10	203.163.247.241	79	1937	0.012999	8.318743				[Errno 1] Unknown host
11	66.249.73.135	77	2014	0.012670	7.331414	crawl-66-249-73-135.googlebot.com	135.73.249.66.in-addr.arpa	66.249.73.135	
12	194.163.179.92	60	2074	0.009873	2.341287	vmi660635.contaboserver.net	92.179.163.194.in-addr.arpa	194.163.179.92	
13	66.249.73.137	58	2132	0.009544	0.850831	crawl-66-249-73-137.googlebot.com	137.73.249.66.in-addr.arpa	66.249.73.137	
14	109.70.100.30	58	2190	0.009544	0.860375	tor-exit-anonymizer.appliedprivacy.net	30.100.70.109.in-addr.arpa	109.70.100.30	

Add a new `hostname` column, by matching IP adresses to their hostnames.

```
ip_host_dict = dict(zip(host_df['ip_address'], host_df['hostname']))
logs_df['hostname'] = [ip_host_dict[ip] for ip in logs_df['client']]
```

Split the request URLs into their components.

```
request_url_df = adv.url_to_df(logs_df['request'])
request_url_df = request_url_df.add_prefix('request_')
request_url_df.head(10)
```

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
0	/			/			nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
1	/.env			/.env			nan	nan	.env	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	.env	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
2	/			/			nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
3	/robots.txt			/robots.txt			nan	nan	robots.txt	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	robots.txt	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
4	/			/			nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
5	/login			/login			nan	nan	login	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	login	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
6	/robots.txt			/robots.txt			nan	nan	robots.txt	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	robots.txt	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
7	/staging/urlytics/			/staging/urlytics/			nan	nan	staging	urlytics	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	urlytics	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
8	/staging/urlytics/_dash-component-suites/dash/html/dash_html_components.v2_0_0m1638886228.min.js			/staging/urlytics/_dash-component-suites/dash/html/dash_html_components.v2_0_0m1638886228.min.js			nan	nan	staging	urlytics	_dash-component-suites	dash	html	dash_html_components.v2_0_0m1638886228.min.js	nan	nan	nan	nan	nan	nan	nan	dash_html_components.v2_0_0m1638886228.min.js	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
9	/staging/urlytics/_dash-layout			/staging/urlytics/_dash-layout			nan	nan	staging	urlytics	_dash-layout	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	_dash-layout	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan

Do the same for the URLs in the `referer` column.

```python
referer_url_df = adv.url_to_df(logs_df['referer'])
referer_url_df = referer_url_df.add_prefix('referer_')
referer_url_df.head(10)
```

	referer_url	referer_scheme	referer_netloc	referer_path	referer_query	referer_fragment	referer_hostname	referer_port	referer_dir_1	referer_dir_2	referer_dir_3	referer_last_dir
0	-			-			nan	nan	-	nan	nan	-
1	-			-			nan	nan	-	nan	nan	-
2	http://adver.tools/	http	adver.tools	/			nan	nan	nan	nan	nan	nan
3	-			-			nan	nan	-	nan	nan	-
4	-			-			nan	nan	-	nan	nan	-
5	-			-			nan	nan	-	nan	nan	-
6	-			-			nan	nan	-	nan	nan	-
7	-			-			nan	nan	-	nan	nan	-
8	http://www.adver.tools/staging/urlytics/	http	www.adver.tools	/staging/urlytics/			nan	nan	staging	urlytics	nan	urlytics
9	http://www.adver.tools/staging/urlytics/	http	www.adver.tools	/staging/urlytics/			nan	nan	staging	urlytics	nan	urlytics

Parse the `user_agent` column.

```python
ua_df = pd.json_normalize([user_agent_parser.Parse(ua) for ua in logs_df['user_agent']])
ua_df.columns = 'ua_' + ua_df.columns.str.replace('user_agent\.', '', regex=True)
ua_df.head(10)
```

	ua_string	ua_family	ua_major	ua_minor	ua_patch	os_family	os_major	os_minor	os_patch	os_patch_minor	device_brand	device_family	device_model
0	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.80 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	Googlebot	2	1		Android	6	0	1		Spider	Spider	Smartphone
1	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36	Chrome	81	0	4044	Linux					Other		
2	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.72 Safari/537.36	Chrome	90	0	4430	Windows	10				Other		
3	advertools/0.13.0	Other				Other					Other		
4	advertools/0.13.0	Other				Other					Other		
5	Mozilla/5.0 zgrab/0.x	Other				Other					Other		
6	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	Googlebot	2	1		Other					Spider	Spider	Desktop
7	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	Googlebot	2	1		Other					Spider	Spider	Desktop
8	Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Googlebot/2.1; +http://www.google.com/bot.html) Chrome/98.0.4758.80 Safari/537.36	Googlebot	2	1		Other					Spider	Spider	Desktop
9	Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Googlebot/2.1; +http://www.google.com/bot.html) Chrome/98.0.4758.80 Safari/537.36	Googlebot	2	1		Other					Spider	Spider	Desktop

Combine all data into one DataFrame and save to a new *.parquet* file.

```
(pd.concat([logs_df, request_url_df, referer_url_df, ua_df], axis=1)
.to_parquet('data/adv_logs_final.parquet', index=False, version='2.4'))
```

Start the analysis.

The advantage of using the parquet format is that the file doens't need to be loaded into memory, and can be queried from disk, just like querying a database. This means you only load the columns that you select, and the rows that satisfy certain conditions. For example we can load the `ua_device.family` and `ua_family` columns, and only the rows where `'ua_device.family', '==', 'Spider'`. We then count the values in the `ua_family` column, and get the top bots accessing our website.

```
top_bots = pd.read_parquet('data/adv_logs_final.parquet',
                filters=[
                    ('ua_device.family', '==', 'Spider')
                ],
                columns=['ua_device.family', 'ua_family'])['ua_family'].value_counts()
top_bots[:15]
```

```
Googlebot      499
PetalBot        46
AhrefsBot       42
Chrome          29
YandexBot       29
LinkedInBot     23
Baiduspider     18
DotBot          17
Twitterbot      16
bingbot         12
MJ12bot         12
Java            10
Nutch            8
masscan          6
FacebookBot      4
Name: ua_family, dtype: int64
```

Happy analyzing!

1.12 Parse and Analyze Crawl Logs in a Dataframe

While crawling with the `crawl()` function, the process produces logs for every page crawled, scraped, redirected, and even blocked by robots.txt rules.

By default, those logs are can be seen on the command line as their default destination is stdout.

A good practice is to set a `LOG_FILE` so you can save those logs to a text file, and review them later. There are several reasons why you might want to do that:

- Blocked URLs: The crawler obeys robots.txt rules by default, and when it encounters pages that it shouldn't crawl, it doesn't. However, this is logged as an event, and you can easily extract a list of blocked URLs from the logs.
- Crawl errors: You might also get some errors while crawling, and it can be interesting to know which URLs generated errors.
- Filtered pages: Those are pages that were discovered but weren't crawled because they are not a sub-domain of the provided url_list, or happen to be on external domains altogether.

This can simply be done by specifying a file name through the optional *custom_settings* parameter of `crawl`:

```
>>> import advertools as adv
>>> adv.crawl('https://example.com',
              output_file='example.jl',
              follow_links=True,
              custom_settings={'LOG_FILE': 'example.log'})
```

If you run it this way, all logs will be saved to the file you chose, *example.log* in this case.

Now, you can use the *crawllogs_to_df()* function to open the logs in a DataFrame:

```
>>> import advertools as adv
>>> logs_df = adv.crawllogs_to_df('example.log')
```

The DataFrame might contain the following columns:

- *time*: The timestamp for the process
- *middleware*: The middleware responsible for this process, whether it is the core engine, the scraper, error handler and so on.
- *level*: The logging level (DEBUG, INFO, etc.)
- *message*: A single word summarizing what this row represents, "Crawled", "Scraped", "Filtered", and so on.
- *domain*: The domain name of filtered (not crawled pages) typically for URLs outside the current website.
- *method*: The HTTP method used in this process (GET, PUT, etc.)
- *url*: The URL currently under process.
- *status*: HTTP status code, 200, 404, etc.
- *referer*: The referring URL, where applicable.
- *method_to*: In redirect rows the HTTP method used to crawl the URL going to.
- *redirect_to*: The URL redirected to.
- *method_from*: In redirect rows the HTTP method used to crawl the URL coming from.
- *redirect_from*: The URL redirected from.
- *blocked_urls*: The URLs that were not crawled due to robots.txt rules.

crawllogs_to_df(*logs_file_path*)

Convert a crawl logs file to a DataFrame.

An interesting option while using the `crawl` function, is to specify a destination file to save the logs of the crawl process itself. This contains additional information about each crawled, scraped, blocked, or redirected URL.

What you would most likely use this for is to get a list of URLs blocked by robots.txt rules. These can be found und the column `blocked_urls`. Crawling errors are also interesting, and can be found in rows where `message` is equal to "error".

```
>>> import advertools as adv
>>> adv.crawl('https://example.com',
              output_file='example.jl',
              follow_links=True,
              custom_settings={'LOG_FILE': 'example.log'})
>>> logs_df = adv.crawl_logs_to_df('example.log')
```

Parameters **logs_file_path** (*str*) -- The path to the logs file.

Returns DataFrame crawl_logs_df A DataFrame summarizing the logs.

logs_to_df(*log_file*, *output_file*, *errors_file*, *log_format*, *fields=None*)

Parse and compress any log file into a DataFrame format.

Convert a log file to a *parquet* file in a DataFrame format, and save all errors (or lines not conformig to the chosen log format) into a separate `errors_file` text file. Any non-JSON log format is possible, provided you have the right regex for it. A few default ones are provided and can be used. Check out `adv.LOG_FORMATS` and `adv.LOG_FIELDS` for the available formats and fields.

```
>>> import advertools as adv
>>> import pandas as pd
>>> adv.logs_to_df(log_file='access.log',
...                output_file='access_logs.parquet',
...                errors_file='log_errors.csv',
...                log_format='common',
...                fields=None)
>>> logs_df = pd.read_parquet('access_logs.parquet')
```

You can now analyze `logs_df` as a normal pandas DataFrame.

Parameters

- **log_file** (*str*) -- The path to the log file.
- **output_file** (*str*) -- The path to the desired output file. Must have a ".parquet" extension, and must not have the same path as an existing file.
- **errors_file** (*str*) -- The path where the parsing errors are stored. Any text format works, CSV is recommended to easily open it with any CSV reader with the separator as "@@".
- **log_format** (*str*) -- Either the name of one of the supported log formats, or a regex of your own format.
- **fields** (*str*) -- A list of fields, which will become the names of columns in `output_file`. Only required if you provide a custom (regex) `log_format`.

1.13 Reverse DNS Lookup in Bulk

Getting the host name of a list of IP addresses can be useful in verifying the authenticity of those IP addresses. You typically want to do this as part of a *log file analysis* pipeline. In this case you have requests made to your server claiming to be of a certain user agent/bot name. Performing a *reverse_dns_lookup()* on those IP addresses, will get you the actual host name that they belong to.

What the *reverse_dns_lookup()* function does, is simply like running the *host* command from the command line, but on a massive scale:

```
$ host 66.249.80.0
0.80.249.66.in-addr.arpa domain name pointer google-proxy-66-249-80-0.google.com.
```

Because you usually have a large number of duplicated IP addresses that you want to check, this function makes the process practical and efficient, in comparison to running the command thousands of times from the comand line.

Running the function is very simple, you simply supply a list of the IP addresses that you have. Make sure to **keep the duplicates**, because the function handles that for you, as well as provide counts and some statistics on the frequency of the IPs:

```
import advertools as adv
ip_list = ['66.249.66.194', '66.249.66.194', '66.249.66.194',
        '66.249.66.91', '66.249.66.91', '130.185.74.243',
        '31.56.96.51', '5.211.97.39']

host_df = adv.reverse_dns_lookup(ip_list)
host_df
```

	ip_address	count	cum_count	perc	cum_perc	hostname	aliaslist	ipaddrlist	errors
0	66.249.66.194	3	3	0.375	0.375	crawl-66-249-66-194.googlebot.com	194.66.249.66.in-addr.arpa	66.249.66.194	
1	66.249.66.91	2	5	0.25	0.625	crawl-66-249-66-91.googlebot.com	91.66.249.66.in-addr.arpa	66.249.66.91	
2	130.185.74.243	1	6	0.125	0.75	mail.garda.ir	243.74.185.130.in-addr.arpa	130.185.74.243	
3	31.56.96.51	1	7	0.125	0.875	31-56-96-51.shatel.ir	51.96.56.31.in-addr.arpa	31.56.96.51	
4	5.211.97.39	1	8	0.125	1				[Errno 1] Unknown host

As you can see, in addition to getting hostnames, aliaslist, and ipaddrlist for the IPs you supplied, you also get counts (absolute and cumulative) as well as percentages (absolute and cumulative). This can give you a good overview of the relative importance of each IP, and can help focus attention on them as needed.

reverse_dns_lookup(*ip_list*, *max_workers=600*)

Return the hostname, aliaslist, and ipaddrlist for a list of IP addresses.

This is mainly useful for a long list of typically duplicated IP adresses and helps in getting the information very fast. It is basically the equivalent of running the *host* command on the command line many times:

```
$ host 66.249.80.0
0.80.249.66.in-addr.arpa domain name pointer google-proxy-66-249-80-0.google.com.
```

You also get a simple report about the counts of the IPs to get an overview of the top ones:

```
>>> import advertools as adv
>>> ip_list = ['66.249.66.194', '66.249.66.194', '66.249.66.194',
...            '66.249.66.91', '66.249.66.91', '130.185.74.243',
...            '31.56.96.51', '5.211.97.39']
>>> adv.reverse_dns_lookup([ip_list])
```

	ip_address	count	cum_count	perc	cum_perc	hostname	aliaslist	ipad-drlist	errors
0	66.249.66.194	3	3	0.375	0.375	crawl-66-249-66-194.googlebot.com	194.66.249.66.in-addr.arpa	66.249.66.194	
1	66.249.66.91	5	5	0.25	0.625	crawl-66-249-66-91.googlebot.com	91.66.249.66.in-addr.arpa	66.249.66.91	
2	130.185.74.243	6	6	0.125	0.75	mail.garda.ir	243.74.185.130.in-addr.arpa	130.185.74.243	
3	31.56.96.51	7	7	0.125	0.875	31-56-96-51.shatel.ir	51.96.56.31.in-addr.arpa	31.56.96.51	
4	5.211.97.39	8	8	0.125	1				[Errno 1] Unknown host

Parameters

- **ip_list** (*list*) -- a list of IP addresses.
- **max_workers** (*int*) -- The maximum number of workers to use for multi processing.

1.14 Import Search Engine Results Pages (SERPs) for Google and YouTube

Analyzing a single SERP is like getting one person to fill out a questionnaire and calling it a survey.

Just like surveys, SERPs need to be collected in large-enough numbers that are representative of the industry/market you want to understand. This is the main feature of the `serp_` functions. They allow you to get the SERPs for a list of queries, across several dimensions (like country, search type, start position, and so on).

There are many parameters that can be used, and you can supply a list for each. The function will get the SERPs for the *product* of all those lists. For example, let's say you you provide the following arguments to the *serp_goog()* function:

- *q*: ['serp tools', 'best serp tools', 'serp tool reviews']
- *gl*: ['us', 'ca', 'uk', 'au', 'nz']
- *start*: [1, 11, 21]

The function will produce: 3 (queries) x 5 (countries) x 3 (start positions) = 45 requests

You typically get ten results each, so in this case you would get 450 rows of data.

All this is done in with one line of code. The result is a single DataFrame with a row for each result, and columns for each attribute (title, snippet, etc.), as well as meta data columns, like *queryTime* and the parameters you selected (*q*, *gl*, and *start* in this case).

Before being able to run queries using *serp_goog()*, you will need to set up some credentials as follows (you don't need a custom search engine for *serp_youtube()*):

- Create a custom search engine: At first, you might be asked to enter a site to search. Enter any domain, then go to the control panel and remove it. Make sure you enable "Search the entire web" and image search. You will also need to get your search engine ID, which you can find on the control panel page.
- Enable the custom search API: The service will allow you to retrieve and display search results from your custom search engine programmatically. You will need to create a project for this first.

- Create credentials for this project: so you can get your key.
- Enable billing for your project if you want to run more than 100 queries per day. The first 100 queries are free; then for each additional 1,000 queries, you pay $5.

serp_goog(*q*, *cx*, *key*, *c2coff=None*, *cr=None*, *dateRestrict=None*, *exactTerms=None*, *excludeTerms=None*, *fileType=None*, *filter=None*, *gl=None*, *highRange=None*, *hl=None*, *hq=None*, *imgColorType=None*, *imgDominantColor=None*, *imgSize=None*, *imgType=None*, *linkSite=None*, *lowRange=None*, *lr=None*, *num=None*, *orTerms=None*, *relatedSite=None*, *rights=None*, *safe=None*, *searchType=None*, *siteSearch=None*, *siteSearchFilter=None*, *sort=None*, *start=None*)

Query Google and get search results in a DataFrame.

For each parameter, you can supply single or multiple values / arguments. If you pass multiple arguments, all the possible combinations of arguments (the product) will be requested, and you will get one DataFrame combining all queries. See examples below.

Parameters

- **q** -- The search expression.
- **cx** -- The custom search engine ID to use for this request.
- **key** -- The API key of your custom search engine.
- **c2coff** -- Enables or disables Simplified and Traditional Chinese Search. The default value for this parameter is 0 (zero), meaning that the feature is enabled. Supported values are:1: Disabled0: Enabled (default)
- **cr** -- Restricts search results to documents originating in a particular country. You may use Boolean operators in the cr parameter's value.Google Search determines the country of a document by analyzing:the top- level domain (TLD) of the document's URLthe geographic location of the Web server's IP addressSee the Country Parameter Values page for a list of valid values for this parameter.
- **dateRestrict** -- Restricts results to URLs based on date. Supported values include:d[number]: requests results from the specified number of past days. - d[number]: requests results from the specified number of past days. - w[number]: requests results from the specified number of past weeks. - m[number]: requests results from the specified number of past months. - y[number]: requests results from the specified number of past years.
- **exactTerms** -- Identifies a phrase that all documents in the search results must contain.
- **excludeTerms** -- Identifies a word or phrase that should not appear in any documents in the search results.
- **fileType** -- Restricts results to files of a specified extension. A list of file types indexable by Google can be found in Search Console Help Center.
- **filter** -- Controls turning on or off the duplicate content filter.See Automatic Filtering for more information about Google's search results filters. Note that host crowding filtering applies only to multi-site searches.By default, Google applies filtering to all search results to improve the quality of those results. Acceptable values are: "0": Turns off duplicate content filter. "1": Turns on duplicate content filter.
- **gl** -- Geolocation of end user. The gl parameter value is a two-letter country code. The gl parameter boosts search results whose country of origin matches the parameter value. See the Country Codes page for a list of valid values.Specifying a gl parameter value should lead to more relevant results. This is particularly true for international customers and, even more specifically, for customers in English- speaking countries other than the United States.

- **highRange** -- Specifies the ending value for a search range.Use lowRange and highRange to append an inclusive search range of lowRange...highRange to the query.
- **hl** -- Sets the user interface language. Explicitly setting this parameter improves the performance and the quality of your search results.See the Interface Languages section of Internationalizing Queries and Results Presentation for more information, and Supported Interface Languages for a list of supported languages.
- **hq** -- Appends the specified query terms to the query, as if they were combined with a logical AND operator.
- **imgColorType** -- Returns black and white, grayscale, or color images: mono, gray, and color. Acceptable values are: "color": color "gray": gray "mono": mono
- **imgDominantColor** -- Returns images of a specific dominant color. Acceptable values are: "black": black "blue": blue "brown": brown "gray": gray "green": green "orange": orange "pink": pink "purple": purple "red": red "teal": teal "white": white "yellow": yellow
- **imgSize** -- Returns images of a specified size. Acceptable values are: "huge": huge "icon": icon "large": large "medium": medium "small": small "xlarge": xlarge "xxlarge": xxlarge
- **imgType** -- Returns images of a type. Acceptable values are: "clipart": clipart "face": face "lineart": lineart "news": news "photo": photo
- **linkSite** -- Specifies that all search results should contain a link to a particular URL
- **lowRange** -- Specifies the starting value for a search range. Use lowRange and highRange to append an inclusive search range of lowRange...highRange to the query.
- **lr** -- Restricts the search to documents written in a particular language (e.g., lr=lang_ja). Acceptable values are: "lang_ar": Arabic "lang_bg": Bulgarian "lang_ca": Catalan "lang_cs": Czech "lang_da": Danish "lang_de": German "lang_el": Greek "lang_en": English "lang_es": Spanish "lang_et": Estonian "lang_fi": Finnish "lang_fr": French "lang_hr": Croatian "lang_hu": Hungarian "lang_id": Indonesian "lang_is": Icelandic "lang_it": Italian "lang_iw": Hebrew "lang_ja": Japanese "lang_ko": Korean "lang_lt": Lithuanian "lang_lv": Latvian "lang_nl": Dutch "lang_no": Norwegian "lang_pl": Polish "lang_pt": Portuguese "lang_ro": Romanian "lang_ru": Russian "lang_sk": Slovak "lang_sl": Slovenian "lang_sr": Serbian "lang_sv": Swedish "lang_tr": Turkish "lang_zh- CN": Chinese (Simplified) "lang_zh-TW": Chinese (Traditional)
- **num** -- Number of search results to return.Valid values are integers between 1 and 10, inclusive.
- **orTerms** -- Provides additional search terms to check for in a document, where each document in the search results must contain at least one of the additional search terms.
- **relatedSite** -- Specifies that all search results should be pages that are related to the specified URL.
- **rights** -- Filters based on licensing. Supported values include: cc_publicdomain, cc_attribute, cc_sharealike, cc_noncommercial, cc_nonderived, and combinations of these.
- **safe** -- Search safety level. Acceptable values are: "active": Enables SafeSearch filtering. "off": Disables SafeSearch filtering. (default)
- **searchType** -- Specifies the search type: image. If unspecified, results are limited to webpages. Acceptable values are: "image": custom image search.
- **siteSearch** -- Specifies all search results should be pages from a given site.
- **siteSearchFilter** -- Controls whether to include or exclude results from the site named in the siteSearch parameter. Acceptable values are: "e": exclude "i": include

- **sort** -- The sort expression to apply to the results.
- **start** -- The index of the first result to return.Valid value are integers starting 1 (default) and the second result is 2 and so forth. For example &start=11 gives the second page of results with the default "num" value of 10 results per page.Note: No more than 100 results will ever be returned for any query with JSON API, even if more than 100 documents match the query, so setting (start + num) to more than 100 will produce an error. Note that the maximum value for num is 10.

The following function call will produce two queries: "hotel" in the USA, and "hotel" in France

```
>>> serp_goog(q='hotel', gl=['us', 'fr'], cx='YOUR_CX', key='YOUR_KEY')
```

The below function call will prouce four queries and make four requests:

- "fligts" in UK
- "fligts" in Australia
- "tickets" in UK
- "tickets" in Australia

'cr' here refers to 'country restrict', which focuses on content originating from the specified country.

```
>>> serp_goog(q=['flights', 'tickets'], cr=['countryUK', 'countryAU'],
              cx='YOUR_CX', key='YOUR_KEY')
```

serp_youtube(*key*, *q=None*, *channelId=None*, *channelType=None*, *eventType=None*, *forContentOwner=None*, *forDeveloper=None*, *forMine=None*, *location=None*, *locationRadius=None*, *maxResults=None*, *onBehalfOfContentOwner=None*, *order=None*, *pageToken=None*, *publishedAfter=None*, *publishedBefore=None*, *regionCode=None*, *relatedToVideoId=None*, *relevanceLanguage=None*, *safeSearch=None*, *topicId=None*, *type=None*, *videoCaption=None*, *videoCategoryId=None*, *videoDefinition=None*, *videoDimension=None*, *videoDuration=None*, *videoEmbeddable=None*, *videoLicense=None*, *videoSyndicated=None*, *videoType=None*)

Query the YouTube API and get search results in a DataFrame. For each parameter you can supply a single or multiple value(s). Looping and merging results is handled automatically in case of multiple values.

Parameters

- **q** -- (string) The q parameter specifies the query term to search for. Your request can also use the Boolean NOT (-) and OR (|) operators to exclude videos or to find videos that are associated with one of several search terms. For example, to search for videos matching either "boating" or "sailing", set the q parameter value to boating|sailing. Similarly, to search for videos matching either "boating" or "sailing" but not "fishing", set the q parameter value to boating|sailing -fishing. Note that the pipe character must be URL- escaped when it is sent in your API request. The URL-escaped value for the pipe character is %7C.
- **channelId** -- (string) The channelId parameter indicates that the API response should only contain resources created by the channel. Note: Search results are constrained to a maximum of 500 videos if your request specifies a value for the channelId parameter and sets the type parameter value to video, but it does not also set one of the forContentOwner, forDeveloper, or forMine filters.
- **channelType** -- (string) The channelType parameter lets you restrict a search to a particular type of channel. Acceptable values are:

  any – Return all channels.

  show – Only retrieve shows.

- **eventType** -- (string) The `eventType` parameter restricts a search to broadcast events. If you specify a value for this parameter, you must also set the type parameter's value to video. Acceptable values are:

  completed – Only include completed broadcasts.

  live – Only include active broadcasts.

  upcoming – Only include upcoming broadcasts.

- **forContentOwner** -- (boolean) This parameter can only be used in a properly authorized request, and it is intended exclusively for YouTube content partners. The `forContentOwner` parameter restricts the search to only retrieve videos owned by the content owner identified by the `onBehalfOfContentOwner` parameter. If `forContentOwner` is set to true, the request must also meet these requirements:The `onBehalfOfContentOwner` parameter is required.The user authorizing the request must be using an account linked to the specified content owner. The `type` parameter value must be set to video.None of the following other parameters can be set: `videoDefinition`, `videoDimension`, `videoDuration`, `videoLicense`, `videoEmbeddable`, `videoSyndicated`, `videoType`.

- **forDeveloper** -- (boolean) This parameter can only be used in a properly authorized request. The `forDeveloper` parameter restricts the search to only retrieve videos uploaded via the developer's application or website. The API server uses the request's authorization credentials to identify the developer. The `forDeveloper` parameter can be used in conjunction with optional search parameters like the `q` parameter. For this feature, each uploaded video is automatically tagged with the project number that is associated with the developer's application in the Google Developers Console. When a search request subsequently sets the `forDeveloper` parameter to `true` the API server uses the request's authorization credentials to identify the developer. Therefore, a developer can restrict results to videos uploaded through the developer's own app or website but not to videos uploaded through other apps or sites.

- **forMine** -- (boolean) This parameter can only be used in a properly authorized request. The `forMine` parameter restricts the search to only retrieve videos owned by the authenticated user. If you set this parameter to `true`, then the `type` parameter's value must also be set to `video`. In addition, none of the following other parameters can be set in the same request: `videoDefinition`, `videoDimension`, `videoDuration`, `videoLicense`, `videoEmbeddable`, `videoSyndicated`, `videoType`.

- **relatedToVideoId** -- (string) The `relatedToVideoId` parameter retrieves a list of videos that are related to the video that the parameter `value` identifies. The parameter `value` must be set to a YouTube video ID and, if you are using this parameter, the `type` parameter must be set to video.Note that if the `relatedToVideoId` parameter is set, the only other supported parameters are `part`, `maxResults`, `pageToken`, `regionCode`, `relevanceLanguage`, `safeSearch`, `type` (which must be set to video), and `fields`.

- **location** -- (string) The `location` parameter, in conjunction with the `locationRadius` parameter, defines a circular geographic area and also restricts a search to videos that specify, in their metadata, a geographic location that falls within that area. The parameter value is a string that specifies latitude/longitude coordinates e.g. (37.42307,-122.08427).The location parameter value identifies the point at the center of the area. The `locationRadius` parameter specifies the maximum distance that the location associated with a video can be from that point for the video to still be included in the search results. The API returns an error if your request specifies a value for the `location` parameter but does not also specify a value for the `locationRadius` parameter.

- **locationRadius** -- (string) The `locationRadius` parameter, in conjunction with the `location` parameter, defines a circular geographic area. The parameter value must be a

floating point number followed by a measurement unit. Valid measurement units are m, km, ft, and mi. For example, valid parameter values include 1500m, 5km, 10000ft, and 0.75mi. The API does not support `locationRadius` parameter values larger than 1000 kilometers. Note: See the definition of the `location` parameter for more information.

- **`maxResults`** -- (unsigned integer) The `maxResults` parameter specifies the maximum number of items that should be returned in the result set. Acceptable values are 0 to 50, inclusive. The default value is 5.
- **`onBehalfOfContentOwner`** -- (string) This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The `onBehalfOfContentOwner` parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.
- **`order`** -- (string) The order parameter specifies the method that will be used to order resources in the API response. The default value is relevance. Acceptable values are:

  date – Resources are sorted in reverse chronological order based on the date they were created.

  rating – Resources are sorted from highest to lowest rating.

  relevance – Resources are sorted based on their relevance to the search query. This is the default value for this parameter.

  title – Resources are sorted alphabetically by title.

  videoCount – Channels are sorted in descending order of their number of uploaded videos.

  viewCount – Resources sorted from highest to lowest number of views. For live broadcasts, videos are sorted by number of concurrent viewers while the broadcasts are ongoing.
- **`pageToken`** -- (string) The `pageToken` parameter identifies a specific page in the result set that should be returned. In an API response, the `nextPageToken` and `prevPageToken` properties identify other pages that could be retrieved.
- **`publishedAfter`** -- (datetime) The `publishedAfter` parameter indicates that the API response should only contain resources created at or after the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).
- **`publishedBefore`** -- (datetime) The `publishedBefore` parameter indicates that the API response should only contain resources created before or at the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).
- **`regionCode`** -- (string) The `regionCode` parameter instructs the API to return search results for videos that can be viewed in the specified country. The parameter value is an ISO 3166-1 alpha-2 country code.
- **`relevanceLanguage`** -- (string) The `relevanceLanguage` parameter instructs the API to return search results that are most relevant to the specified language. The parameter value is typically an ISO 639-1 two-letter language code. However, you should use the values zh-Hans for simplified Chinese and zh-Hant for traditional Chinese. Please note that results in other languages will still be returned if they are highly relevant to the search query term.
- **`safeSearch`** -- (string) The `safeSearch` parameter indicates whether the search results should include restricted content as well as standard content. Acceptable values are:

moderate – YouTube will filter some content from search results and, at the least, will filter content that is restricted in your locale. Based on their content, search results could be removed from search results or demoted in search results. This is the default parameter value.

none – YouTube will not filter the search result set.

strict – YouTube will try to exclude all restricted content from the search result set.

Based on their content, search results could be removed from search results or demoted in search results.

- **topicId** -- (string) The `topicId` parameter indicates that the API response should only contain resources associated with the specified topic. The value identifies a Freebase topic ID.
- **type** -- (string) The `type` parameter restricts a search query to only retrieve a particular type of resource. The value is a comma-separated list of resource types. The default value is video,channel,playlist. Acceptable values are: channel, playlist, and video
- **videoCaption** -- (string) The `videoCaption` parameter indicates whether the API should filter video search results based on whether they have captions. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  any – Do not filter results based on caption availability.

  closedCaption – Only include videos that have captions.

  none – Only include videos that do not have captions.
- **videoCategoryId** -- (string) The `videoCategoryId` parameter filters video search results based on their category. If you specify a value for this parameter, you must also set the `type` parameter's value to video.
- **videoDefinition** -- (string) The `videoDefinition` parameter lets you restrict a search to only include either high definition (HD) or standard definition (SD) videos. HD videos are available for playback in at least 720p, though higher resolutions, like 1080p, might also be available. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  any – Return all videos, regardless of their resolution.

  high – Only retrieve HD videos.

  standard – Only retrieve videos in standard definition.
- **videoDimension** -- (string) The `videoDimension` parameter lets you restrict a search to only retrieve 2D or 3D videos. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  2d – Restrict search results to exclude 3D videos.

  3d – Restrict search results to only include 3D videos.

  any – Include both 3D and non-3D videos in returned results. This is the default value.
- **videoDuration** -- (string) The `videoDuration` parameter filters video search results based on their duration. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  any – Do not filter video search results based on their duration. This is the default value.

  long – Only include videos longer than 20 minutes.

  medium – Only include videos that are between four and 20 minutes long (inclusive).

short – Only include videos that are less than four minutes long.

- **videoEmbeddable** -- (string) The `videoEmbeddable` parameter lets you to restrict a search to only videos that can be embedded into a webpage. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  any – Return all videos, embeddable or not.

  true – Only retrieve embeddable videos.

- **videoLicense** -- (string) The `videoLicense` parameter filters search results to only include videos with a particular license. YouTube lets video uploaders choose to attach either the Creative Commons license or the standard YouTube license to each of their videos. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  any – Return all videos, regardless of which license they have, that match the query parameters.

  creativeCommon – Only return videos that have a Creative Commons license. Users can reuse videos with this license in other videos that they create.

  youtube – Only return videos that have the standard YouTube license.

- **videoSyndicated** -- (string) The `videoSyndicated` parameter lets you to restrict a search to only videos that can be played outside youtube.com. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  any – Return all videos, syndicated or not.

  true – Only retrieve syndicated videos.

- **videoType** -- (string) The `videoType` parameter lets you restrict a search to a particular type of videos. If you specify a value for this parameter, you must also set the `type` parameter's value to video. Acceptable values are:

  any – Return all videos.

  episode – Only retrieve episodes of shows.

  movie – Only retrieve movies.

set_logging_level(*level_or_name*)

Change the logging level during the session. Acceptable values are [0, 10, 20, 30, 40, 50, 'NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL']

youtube_channel_details(*key*, *channel_ids*)

Return details of channels for which the ids are given. Assumes `ids` is a comma-separated list of channel ids with no spaces.

youtube_video_details(*key*, *vid_ids*)

Return details of videos for which the ids are given. Assumes `ids` is a comma-separated list of video ids with no spaces.

1.15 Import and Analyze Knowledge Graph Results on a Large Scale

If *analyzing SERPs* is the first step in understanding your rankings on search engines, then analyzing the knowledge graph can be thought of as step zero.

SERP positions for a certain keyword show how each page is ranked in comparison to all other eligible pages. Knowledge graph scores on the other hand, show the ranks of the different meanings that a word can take for Google (a person, a city, a brand, etc.).

> **Warning:** From Google's documentation: This API is not suitable for use as a production-critical service. Your product should not form a critical dependence on this API.

It's not clear whether this is from a technical reliability or a content correctness point of view, but it is what the docs mention. So please keep this in mind when using it.

1.15.1 Account Setup

In order to be able to send requests, you will need to create a project, set up billing, and activate the knowledge graph API for your project. You will then need to create credentials (API Key). Once you have that, you can use it as your `key` parameter when running requests, as shown below.

1.15.2 How to use Google's Knowledge Graph API

What is "google"? Is it a search engine, a company, a brand, a very large number? What else is it?

And if it is all of those things, what is the relative ranking of each? What is the source of the information, its URL, images (if any)?

```
>>> key = 'YOUR_GOOGLE_DEVELOPER_KEY'
>>> google =  knowledge_graph(key=key, query='google')
>>> google
     query  resultScore                                    result.@type                ␣
↪result.description    result.name
0   google       203191        ['Corporation', 'Organization', 'Thing']                ␣
↪Technology company         Google
1   google        49462                           ['WebSite', 'Thing']                ␣
↪            Google         Search
2   google        19142                           ['WebSite', 'Thing']                ␣
↪               nan          Gmail
3   google        13251                  ['Brand', 'WebSite', 'Thing']                ␣
↪           Website    Google Maps
4   google         7549     ['WebSite', 'SoftwareApplication', 'Thing']                ␣
↪           Website   Google Drive
5   google         6853                           ['WebSite', 'Thing']                ␣
↪           Website    Google Play
6   google         6543               ['SoftwareApplication', 'Thing']                ␣
↪       Web browser  Google Chrome
7   google         4312        ['Corporation', 'Organization', 'Thing']  Multinational␣
↪conglomerate company  Alphabet Inc.
8   google         3395               ['SoftwareApplication', 'Thing']                ␣
↪               nan Google Account
```

(continues on next page)

(continued from previous page)

```
9   google         1306                                    ['Thing']                ␣
↪               nan         Google

>>> google.columns
Index(['query', 'resultScore', '@type', 'result.@type', 'result.description',
   'result.image.contentUrl', 'result.image.url',
   'result.detailedDescription.articleBody',
   'result.detailedDescription.url', 'result.detailedDescription.license',
   'result.url', 'result.name', 'result.@id', 'query_time'],
  dtype='object')
```

The above table is a sample response from the *knowledge_graph()* function. Many more columns are available as you can see in the second line above. We can see that "google" is a company, with a result score of 203,191 and it is a search engine/website with a result score of 49,462. It is then understood as an email application, a mapping application, and so on, as you can see in the *result.name* column.

You can also see that we get the types under which this result falls, in the *result.@type* column. Multiple types show the type inheritance, and as you can also see, everything is a "Thing". This is the top element in the type hierarchy under which everything belongs.

Like the *Google SERP* and *YouTube SERP*, functions this funcion works in the same manner, creating, sending, and aggregating the product of the arguments passed to it.

For example if you run

```
>>> knowledge_graph(key=key, query=['google', 'bing'], languages=['en', 'fr', 'de'])
```

The function will send 2 (queries) x 3 languages = 6 requests.

(google, en), (google, fr), (google, de) , (bing, en), (bing, fr), (bing, de)

This is actually the main value of having this function, because you usually want a large sample to evaluate certain keywords across languages or types.

Let's check what "seo" and "search engine optimization" mean in different languages.

```
>>> seo = knowledge_graph(key=key, query=['seo', 'search engine optimization'],␣
↪languages=['en', 'es', 'de'])
>>> seo
        query                          languages       resultScore         result.name ␣
↪                    result.@type                                          result.
↪description
0       search engine optimization      de                    3587        ␣
↪Suchmaschinenoptimierung            ['Thing']                                        ␣
↪nan
1       search engine optimization      de                     321         Lokale␣
↪Suchmaschinenoptimierung      ['Thing']                                        nan
2       search engine optimization      de                     252         ␣
↪Suchmaschinenmarketing              ['Thing']                                        ␣
↪nan
4       search engine optimization      en                   71756         Search␣
↪engine optimization           ['Thing']                                        nan
5       search engine optimization      en                    5056         Search␣
↪engine marketing              ['Thing']                                        nan
6       search engine optimization      en                     576         SEOP, Inc.  ␣
↪                        ['Organization', 'Corporation', 'Thing']          Company
```

(continues on next page)

(continued from previous page)

```
13      seo                              de                     3313         Seoul
↪                        ['AdministrativeArea', 'Thing', 'City', 'Place']  Hauptstadt
↪von Südkorea
14      seo                              de                     1509         Seo Yea-ji
↪                        ['Thing', 'Person']
↪Schauspielerin
15      seo                              de                      584
↪Suchmaschinenoptimierung             ['Thing']
↪nan
33      seo                              es                     1509         Seo Ye-ji
↪                        ['Person', 'Thing']                               Actriz
34      seo                              es                      584
↪Posicionamiento en buscadores        ['Thing']
↪nan
35      seo                              es                      316         Jin
↪                        ['Person', 'Thing']                               Cantante
53      seo                              en                     8760         Search
↪engine optimization           ['Thing']                                        nan
54      seo                              en                     3313         Seoul
↪                        ['AdministrativeArea', 'Thing', 'City', 'Place']  Capital of
↪South Korea
55      seo                              en                     1435         Sulli
↪                        ['Thing', 'Person']                               South Korean
↪actress
```

```
>>> seo.columns
Index(['query', 'languages', 'resultScore', '@type', 'result.name',
       'result.@type', 'result.@id', 'result.image.contentUrl',
       'result.image.url', 'result.detailedDescription.license',
       'result.detailedDescription.url',
       'result.detailedDescription.articleBody', 'result.description',
       'result.url', 'query_time'],
      dtype='object')
```

It's interesting to see how the same word can mean different things in different contexts.

knowledge_graph(*key*, *query=None*, *ids=None*, *languages=None*, *types=None*, *prefix=None*, *limit=None*)

Query Google's Knowledge Graph with any combination of parameters.

Note that Google's documentation states that "This API is not suitable for use as a production-critical service." So please keep this in mind.

Parameters

- **key** (*string*) -- Your Google developer key.
- **query** (*string*) -- A literal string to search for in the Knowledge Graph.
- **ids** (*string*) -- A list of entity IDs to search for in the Knowledge Graph.
- **languages** (*string*) -- The list of language codes (defined in ISO 639) to run the query with, for instance *en*.
- **types** (*string*) -- Restricts returned entities to those of the specified types. For example, you can specify *Person* (as defined in http://schema.org/Person) to restrict the results to entities representing people. If multiple types are specified, returned entities will contain one

or more of these types.

- **prefix** (*boolean*) -- Enables prefix (initial substring) match against names and aliases of entities. For example, a prefix *Jung* will match entities and aliases such as *Jung*, *Jungle*, and *Jung-ho Kang*.
- **limit** (*number*) -- Limits the number of entities to be returned. Maximum is 500. Default is 20. Requests with high limits have a higher chance of timing out.

https://developers.google.com/knowledge-graph/reference/rest/v1

1.16 Split, Parse, and Analyze URL Structure

Extracting information from URLs can be a little tedious, yet very important. Using the standard for URLs we can extract a lot of information in a fairly structured manner.

There are many situations in which you have many URLs that you want to better understand:

- **Analytics reports**: Whichever analytics sytem you use, whether Google Analytics, search console, or any other reporting tool that reports on URLs, your reports can be enhanced by splitting URLs, and in effect becoming four or five data points as opposed to one.
- *Crawl datasets*: The result of any crawl you run typically contains the URLs, which can benefit from the same enhancement.
- *SERP datasets*: Which are basically about URLs.
- *Extracted URLs*: Extracting URLs from social media posts is one thing you might want to do to better understand those posts, and further splitting URLs can also help.
- *XML sitemaps*: Right after downloading a sitemap(s) splitting it further can help in giving a better perspective on the dataset.

The main function here is *url_to_df()*, which as the name suggests, converts URLs to DataFrames.

```
import advertools as adv

urls = ['https://netloc.com/path_1/path_2?price=10&color=blue#frag_1',
        'https://netloc.com/path_1/path_2?price=15&color=red#frag_2',
        'https://netloc.com/path_1/path_2/path_3?size=sm&color=blue#frag_1',
        'https://netloc.com/path_1?price=10&color=blue']
adv.url_to_df(urls)
```

	url	scheme	net-loc	path	query	frag-ment	dir_1	dir_2	dir_3	last_dir	query_color	query_price	query_size
0	https://netloc.com/path_1/path_2?price=10&color=blue#frag_1	https	net-loc.com	/path_1/path_2	price=10&color=blue	frag_1	path_1	path_2	nan	path_2	blue	10	nan
1	https://netloc.com/path_1/path_2?price=15&color=red#frag_2	https	net-loc.com	/path_1/path_2	price=15&color=red	frag_2	path_1	path_2	nan	path_2	red	15	nan
2	https://netloc.com/path_1/path_2/path_3?size=sm&color=blue#frag_1	https	net-loc.com	/path_1/path_2/path_3	size=sm&color=blue	frag_1	path_1	path_2	path_3	path_3	blue	nan	sm
3	https://netloc.com/path_1?price=10&color=blue	https	net-loc.com	/path_1	price=10&color=blue		path_1	nan	nan	path_1	blue	10	nan

A more elaborate exmaple on *how to analyze URLs* shows how you might use this function after obtaining a set of URLs.

- **url**: The original URLs are listed as a reference. They are decoded for easier reading, and you can set `decode=False` if you want to retain the original encoding.
- **scheme**: Self-explanatory. Note that you can also provide relative URLs */category/sub-category?one=1&two=2* in which case the *url*, *scheme* and *netloc* columns would be empty. You can mix relative and absolute URLs as well.
- **netloc**: The network location is the sub-domain (optional) together with the domain and top-level domain and/or the country domain.
- **path**: The slug of the URL, excluding the query parameters and fragments if any. The path is also split into directories `dir_1/dir_2/dir_3/...` to make it easier to categorize and analyze the URLs.
- **last_dir**: The last directory of each of the URLs. This is usually the part that contains information about the page itself (blog post title, product name, etc.) with previous directories providing meta data (category, sub-category, author name, etc.). In many cases you don't have all URLs with the same number of directories, so they end up unaligned. This extracts all ``last_dir``s in one column.
- **query**: If query parameters are available they are given in this column, but more importantly they are parsed and included in separate columns, where each parameter has its own column (with the keys being the names). As in the example above, the query *price=10&color=blue* becomes two columns, one for price and the other for color. If any other URLs in the dataset contain the same parameters, their values will be populated in the same column, and *NA* otherwise.
- **fragment**: The final part of the URL after the hash mark *#*, linking to a part in the page.
- **query_***: The query parameter names are prepended with *query_* to make it easy to filter them out, and to avoid any name collissions with other columns, if some URL contains a query parameter called "url" for example. In the unlikely event of having a repeated parameter in the same URL, then their values would be delimited by two "@" signs *one@@two@@three*. It's unusual, but it happens.
- **hostname and port**: If available a column for ports will be shown, and if the hostname is different from *netloc* it would also have its own column.

1.16.1 Query Parameters

The great thing about parameters is that the names are descriptive (mostly!) and once given a certain column you can easily understand what data they contain. Once this is done, you can sort the products by price, filter by destination, get the red and blue items, and so on.

1.16.2 The URL Path (Directories):

Here things are not as straightforward, and there is no way to know what the first or second directory is supposed to indicate. In general, I can think of three main situations that you can encounter while analyzing directories.

- **Consistent URLs**: This is the simplest case, where all URLs follow the same structure. */en/product1* clearly shows that the first directory indicates the language of the page. So it can also make sense to rename those columns once you have discovered their meaning.
- **Inconsistent URLs**: This is similar to the previous situation. All URLs follow the same pattern with a few exceptions. Take the following URLs for example:
  - /topic1/title-of-article-1
  - /es/topic1/title-of-article-2
  - /es/topic2/title-of-article-3
  - /topic2/title-of-artilce-4

  You can see that they follow the pattern */language/topic/article-title*, except for English, which is not explicitly mentioned, but its articles can be identified by having two instead of three directories, as we have for "/es/". If URLs are split in this case, yout will end up with *dir_1* having "topic1", "es", "es", and "topic2", which distorts the data. Actually you want to have "en", "es", "es", "en". In such cases, after making sure you have the right rules and patterns, you might create special columns or replace/insert values to make them consistent, and get them to a state similar to the first example.
- **URLs of different types**: In many cases you will find that sites have different types of pages with completely different roles on the site.
  - /blog/post-1-title.html
  - /community/help/topic_1
  - /community/help/topic_2

  Here, once you split the directories, you will see that they don't align properly (because of different lengths), and they can't be compared easily. A good approach is to split your dataset into one for blog posts and another for community content for example.

The ideal case for the *path* part of the URL is to be split into directories of equal length across the dataset, having the right data in the right columns and *NA* otherwise. Or, splitting the dataset and analyzing separately.

`url_to_df`(*urls*, *decode=True*)

Split the given URLs into their components to a DataFrame.

Each column will have its own component, and query parameters and directories will also be parsed and given special columns each.

Parameters

- **urls** (*url*) -- A list of URLs to split into components
- **decode** (*bool*) -- Whether or not to decode the given URLs

Return DataFrame split A DataFrame with a column for each component

1.17 Emoji: Extract, Analyze, and Get Insights

An emoji is worth a thousand words! Regular expressions and helper functionality to aid in extracting and finding emoji from text.

EMOJI_ENTRIES	A dictionary of the full emoji list together with unicode code points, textual name, group, and sub-group. Based on v13.1 https://unicode.org/Public/emoji/13.1/emoji-test.txt
emoji_df	The same dictionary as a pandas DataFrame
extract_emoji()	A function for extracting and summarizing emoji in a text list, with statistics about frequencies and usage.
emoji_search()	A function for searching across names, groups, and sub-groups to find emoji based on your keywords of choice.
EMOJI_RAW	A regular expression to extract the full list. See here on how it was developed: https://www.kaggle.com/eliasdabbas/how-to-create-a-python-regex-to-extract-emoji

1.17.1 Emoji Search

You can search the whole emoji database with the *emoji_search()* function:

```
import advertools as adv

vegetable_emoji = adv.emoji_search('vegetable')
vegetable_emoji.head()
```

	codepoint	status	emoji	name	group	sub_group
0	1F951	fully-qualified		avocado	Food & Drink	food-vegetable
1	1F346	fully-qualified		eggplant	Food & Drink	food-vegetable
2	1F954	fully-qualified		potato	Food & Drink	food-vegetable
3	1F955	fully-qualified		carrot	Food & Drink	food-vegetable
4	1F33D	fully-qualified		ear of corn	Food & Drink	food-vegetable

Keep in mind that the search uses regular expression, and results might not be exactly what you expect.

```
love_emoji = adv.emoji_search('love')
love_emoji
```

	codepoint	status	emoji	name	group	sub_group
0	1F48C	fully-qualified		love letter	Smileys & Emotion	emotion
1	1F91F	fully-qualified		love-you gesture	People & Body	hand-fingers-partial
2	1F91F 1F3FB	fully-qualified		love-you gesture: light skin tone	People & Body	hand-fingers-partial
3	1F91F 1F3FC	fully-qualified		love-you gesture: medium-light skin tone	People & Body	hand-fingers-partial
4	1F91F 1F3FD	fully-qualified		love-you gesture: medium skin tone	People & Body	hand-fingers-partial
5	1F91F 1F3FE	fully-qualified		love-you gesture: medium-dark skin tone	People & Body	hand-fingers-partial
6	1F91F 1F3FF	fully-qualified		love-you gesture: dark skin tone	People & Body	hand-fingers-partial
7	1F340	fully-qualified		four leaf clover	Animals & Nature	plant-other
8	1F3E9	fully-qualified		love hotel	Travel & Places	place-building
9	1F94A	fully-qualified		boxing glove	Activities	sport
10	1F9E4	fully-qualified		gloves	Objects	clothing
11	1F1F8 1F1EE	fully-qualified		flag: Slovenia	Flags	country-flag

1.17.2 Extract Emoji from Text

Many times you might have some social media text, or any regular text containing emoji that you want to analyze. The *extract_emoji()* function does that, and returns useful information about the extracted emoji. You can play around with the following sample text list, modify it, and explore the different stats, and information about the extracted emoji:

```
text_list = ['I feel like playing basketball ',
             'I like playing football ',
             'Not feeling like sports today']

emoji_summary = adv.extract_emoji(text_list)
print(emoji_summary.keys())
```

emoji_search(*regex*)

Return a DataFrame of all emoji entries that match `regex`.

The search is run on the name of the emoji, its group, and sub-group.

Parameters **regex** (*str*) -- regular expression (case insensitive)

```
>>> import advertools as adv
>>> adv.emoji_search('dog')
         codepoint           status  emoji          name             group       ␣
↪sub_group
0             1F436  fully-qualified           dog face  Animals & Nature    animal-
↪mammal
```

(continues on next page)

(continued from previous page)

```
1             1F415  fully-qualified               dog  Animals & Nature    animal-
↪mammal
2             1F9AE  fully-qualified         guide dog  Animals & Nature    animal-
↪mammal
3  1F415 200D 1F9BA  fully-qualified       service dog  Animals & Nature    animal-
↪mammal
4             1F32D  fully-qualified           hot dog      Food & Drink    food-
↪prepared
```

```
>>> blue = adv.emoji_search('blue')
>>> blue
  codepoint           status emoji                name               group     sub_
↪group
0     1F499  fully-qualified               blue heart  Smileys & Emotion      ␣
↪emotion
1     1FAD0  fully-qualified             blueberries       Food & Drink    food-
↪fruit
2     1F4D8  fully-qualified               blue book            Objects    book-
↪paper
3     1F535  fully-qualified             blue circle            Symbols    ␣
↪geometric
4     1F7E6  fully-qualified             blue square            Symbols    ␣
↪geometric
5     1F537  fully-qualified       large blue diamond            Symbols    ␣
↪geometric
6     1F539  fully-qualified       small blue diamond            Symbols    ␣
↪geometric
```

extract_emoji(*text_list*)

Return a summary dictionary about emoji in `text_list`

Get a summary of the number of emoji, their frequency, the top ones, and more.

Parameters **text_list** (*list*) -- A list of text strings.

Returns **summary** A dictionary with various stats about emoji

```
>>> posts = ['I am grinning ','A grinning cat ',
...          'hello!  ', 'Just text']
```

```
>>> emoji_summary = extract_emoji(posts)
>>> emoji_summary.keys()
dict_keys(['emoji', 'emoji_text', 'emoji_flat', 'emoji_flat_text',
'emoji_counts', 'emoji_freq', 'top_emoji', 'top_emoji_text',
'top_emoji_groups', 'top_emoji_sub_groups', 'overview'])
```

```
>>> emoji_summary['emoji']
[[''], [''], ['', '', '', '', ''], []]
```

```
>>> emoji_summary['emoji_text']
[['grinning face'], ['grinning cat'], ['grinning face', 'grinning face',
  'grinning face', 'yellow heart', 'yellow heart'], []]
```

A simple extract of emoji from each of the posts. An empty list if none exist

```
>>> emoji_summary['emoji_flat']
['', '', '', '', '', '', '']
```

```
>>> emoji_summary['emoji_flat_text']
['grinning face', 'grinning cat', 'grinning face', 'grinning face',
'grinning face', 'yellow heart', 'yellow heart']
```

All emoji in one flat list.

```
>>> emoji_summary['emoji_counts']
[1, 1, 5, 0]
```

The count of emoji per post.

```
>>> emoji_summary['emoji_freq']
[(0, 1), (1, 2), (5, 1)]
```

Shows how many posts had 0, 1, 2, 3, etc. emoji (number_of_emoji, count)

```
>>> emoji_summary['top_emoji']
[('', 4), ('', 2), ('', 1)]
```

```
>>> emoji_summary['top_emoji_text']
[('grinning face', 4), ('yellow heart', 2),
 ('grinning cat', 1)]
```

```
>>> emoji_summary['top_emoji_groups']
[('Smileys & Emotion', 7)]
```

```
>>> emoji_summary['top_emoji_sub_groups']
[('face-smiling', 4), ('emotion', 2), ('cat-face', 1)]
```

```
>>> emoji_summary['overview']
{'num_posts': 4,
 'num_emoji': 7,
 'emoji_per_post': 1.75,
 'unique_emoji': 3}
```

1.18 Extract structured entities from text lists

Structured entities are pattern matches and not inferred entities. Some example are hashtags, emoji, mentions, questions, and so on. This is in contrast to entity extraction which are inferred from the context of the sentence (people, companies, brands and so on).

All functions start with `extract_` and have a descriptive name for the type of entity that they extract.

There is also a generic `extract` fucntion which powers all others, and it can be used for any other pattern not included. It takes a regular expression, and returns a similar dictionary to the other functions.

1.18.1 Extract Functions

extract()	A generic function that takes a regex to extract any pattern you want
extract_currency()	Currency symbols together with surrounding text for context. This does not include currency abbreviations (USD, EUR, JPY, etc.), only symbols ($, £, €, etc).
advertools.emoji.extract_emoji()	All the emoji database, together with textual names, groups and sub-groups.
extract_exclamations()	Sentences that end with an excalamation mark!
extract_hashtags()	Extract hashtags with descriptive statistics.
extract_intense_words()	Words that contain three or more repeated characters to express an intense feeling (positive or negative), "I looooooovvvvee this thing".
extract_mentions()	User mentions in social media posts. Also useful for network analysis.
extract_numbers()	Any numbers that are included the text list. Included a modifiable list of separators to use (",", ".", "-", etc.).
extract_questions()	Questions included in the text list.
extract_urls()	URls in the text list.
extract_words()	Any arbitrary words that you want extracted. Works in two modes, either the word should fully match the pattern, or as part of a longer word, ("rest" can be matched from "restaurant" or not).

All functions return a dictionary with the entities extracted, along with helpful statistics. Since the entities have different meanings, most of them return additional keys depending on the context.

The recommended way of using:

```
import advertools as adv

text_list = ['This is the first #text.', 'Second #sentence is here.',
             'Hello, how are you?', 'This #sentence is the last #sentence']
hashtag_summary = adv.extract_hashtags(text_list)
hashtag_summary.keys()
```

```
dict_keys(['hashtags', 'hashtags_flat', 'hashtag_counts', 'hashtag_freq',
           'top_hashtags', 'overview'])
```

Now you can start exploring:

```
hashtag_summary
```

```
>>> hashtag_summary['overview']
{'num_posts': 4,
 'num_hashtags': 4,
 'hashtags_per_post': 1.0,
 'unique_hashtags': 2}
```

```
>>> hashtag_summary['hashtags']
[['#text'], ['#sentence'], [], ['#sentence', '#sentence']]
>>> hashtag_summary['hashtags_flat']
['#text', '#sentence', '#sentence', '#sentence']
>>> hashtag_summary['hashtag_counts']
[1, 1, 0, 2]
```

(continues on next page)

(continued from previous page)

```
>>> hashtag_summary['hashtag_freq']
[(0, 1), (1, 2), (2, 1)]
>>> hashtag_summary['top_hashtags']
[('#sentence', 3), ('#text', 1)]
```

Let's explore a proper dataset of tweets, which you can generate using one of the functions in the *twitter API* module.

```
import advertools as adv
import pandas as pd

tweets = pd.read_csv('data/tweets.csv')
print(tweets.shape)
tweets.head()
```

	tweet_text	follow-ers_count
0	@AERIALMAGZC @penguinnyyyyy you won't be afraid if I give you a real reason :D	157
1	Vibing in the office to #Metallica when the boss is on a coffee break #TheOffice https://t.co/U5vdYevvfe	4687
2	I feel like Ann says she likes coffee and then gets drinks that are 99% sugar and 1% coffee https://t.co/HfuBV4v3aY	104
3	A venti iced coffee with four pumps of white mocha, sweet cream and caramel drizzle might just be my new favorite drink. Shout out to TikTok lol	126
4	I was never a coffee person until I had kids. this cup is a life saver. https://t.co/Zo0CnVuiGj	1595
5	Who's excited about our next Coffee Chat? We know we are! We're also adding Representative John Bradford to this lineup to discuss redistricting in the area. You won't want to miss it! RSVP: https://t.co/R3YNJjJCUG Join the meeting: https://t.co/Ho4Kx7ZZ24 https://t.co/KfPdR3hupY	5004
6	he paid for my coffee= husband	165
7	It's nipply outside, and now I side too :) That sounds like blowjob in front of a fire and visit with coffee after :) I'm still out of coffee I could have green tea instead Hahahahahahaha I want to spend the morning pampering you ...	0
8	Good morning I hope everyone has a great Tuesday morning. Enjoy your day and coffee	189
9	@MarvinMilton2 I nearly choked on my coffee	1160

1.18.2 Extract *#hashtags*

```
hashtag_summary = adv.extract_hashtags(tweets['tweet_text'])
hashtag_summary.keys()
```

```
dict_keys(['hashtags', 'hashtags_flat', 'hashtag_counts', 'hashtag_freq',
        'top_hashtags', 'overview'])
```

```
hashtag_summary['overview']
```

```
{'num_posts': 2000,
'num_hashtags': 733,
```

(continues on next page)

(continued from previous page)

```
'hashtags_per_post': 0.3665,
'unique_hashtags': 572}
```

```
[h for h in hashtag_summary['hashtags'] if h][:10]
```

```
hashtag_summary['top_hashtags'][:10]
```

```
hashtag_summary['hashtag_freq']
```

1.18.3 Extract *@mentions*

```
mention_summary = adv.extract_mentions(tweets['tweet_text'])
mention_summary.keys()
```

```
dict_keys(['mentions', 'mentions_flat', 'mention_counts', 'mention_freq',
           'top_mentions', 'overview'])
```

```
mention_summary['overview']
```

```
{'num_posts': 2000,
'num_mentions': 1346,
'mentions_per_post': 0.673,
'unique_mentions': 1132}
```

```
pd.DataFrame(zip(mention_summary['mentions'],
                 mention_summary['mention_counts']),
             columns=['mentions', 'count'])
```

```
[h for h in mention_summary['mentions'] if h][:10]
```

```
mention_summary['top_mentions'][:10]
```

```
mention_summary['mention_freq']
```

1.18.4 Extract Currency *$ ¢ £ ¤ ¥*

```
currency_summary = adv.extract_currency(tweets['tweet_text'])
currency_summary.keys()
```

```
dict_keys(['currency_symbols', 'currency_symbols_flat',
           'currency_symbol_counts', 'currency_symbol_freq',
           'top_currency_symbols', 'overview', 'currency_symbol_names',
           'surrounding_text'])
```

```
currency_summary['overview']
```

```
{'num_posts': 2000,
'num_currency_symbols': 37,
'currency_symbols_per_post': 0.0185,
'unique_currency_symbols': 4}
```

```
currency_summary['top_currency_symbols']
```

```
[text for text in currency_summary['surrounding_text'] if text][:10]
```

```
[sym for sym in currency_summary['currency_symbol_names'] if sym][:10]
```

1.18.5 Extract numbers *1234567890*

```
number_summary = adv.extract_numbers(tweets['tweet_text'])
number_summary.keys()
```

```
dict_keys(['numbers', 'numbers_flat', 'number_counts', 'number_freq',
           'top_numbers', 'overview'])
```

```
number_summary['overview']
```

```
{'num_posts': 2000,
'num_numbers': 1727,
'numbers_per_post': 0.8635,
'unique_numbers': 257}
```

```
number_summary['number_freq']
```

```
pd.DataFrame({
    'numbers': number_summary['numbers'],
    'counts': number_summary['number_counts'],
}).head(20)
```

1.18.6 Extract questions *? ¿ ⁇ ⁉*

```
question_summary = adv.extract_questions(tweets['tweet_text'])
question_summary.keys()
```

```
dict_keys(['question_marks', 'question_marks_flat', 'question_mark_counts',
           'question_mark_freq', 'top_question_marks', 'overview',
           'question_mark_names', 'question_text'])
```

```
question_summary['overview']
```

```
{'num_posts': 2000,
'num_question_marks': 321,
```

(continues on next page)

(continued from previous page)

```
'question_marks_per_post': 0.1605,
'unique_question_marks': 1}
```

```
question_summary['question_text'][:25]
```

```
[[],
 [],
 [],
 [],
 [],
 ["Who's excited about our next Coffee Chat?"],
 [],
 [],
 [],
 [],
 ['@ckaiserjr @perry_ron @LILGUYISBACK Is it okay if the hot water is flavored with␣
↪coffee?'],
 [],
 [],
 [],
 [],
 [],
 [],
 [],
 [],
 [],
 ["You think if you do that you'll loose your followers ???"],
 [],
 [],
 ['maybe more coffee will help?'],
 []]
```

1.18.7 Extract Exclamations *! ¡*

```
exclamation_summary = adv.extract_exclamations(tweets['tweet_text'])
exclamation_summary.keys()
```

```
dict_keys(['exclamation_marks', 'exclamation_marks_flat',
           'exclamation_mark_counts', 'exclamation_mark_freq',
           'top_exclamation_marks', 'overview', 'exclamation_mark_names',
           'exclamation_text'])
```

```
exclamation_summary['overview']
```

```
{'num_posts': 2000,
'num_exclamation_marks': 563,
'exclamation_marks_per_post': 0.2815,
'unique_exclamation_marks': 2}
```

```
exclamation_summary['top_exclamation_marks']
```

```
exclamation_summary['exclamation_text'][:15]
```

1.18.8 Extract Emoji

```
emoji_summary = adv.extract_emoji(tweets['tweet_text'])
emoji_summary.keys()
```

```
dict_keys(['emoji', 'emoji_text', 'emoji_flat', 'emoji_flat_text',
           'emoji_counts', 'emoji_freq', 'top_emoji', 'top_emoji_text',
           'top_emoji_groups', 'top_emoji_sub_groups', 'overview'])
```

```
emoji_summary['overview']
```

```
{'num_posts': 2000,
'num_emoji': 1149,
'emoji_per_post': 0.5745,
'unique_emoji': 279}
```

```
pd.DataFrame({
    'emoji': emoji_summary['emoji'],
    'emoji_name': emoji_summary['emoji_text']
})[:20]
```

```
emoji_summary['top_emoji'][:20]
```

```
[('', 159),
 ('', 72),
 ('', 64),
 ('', 49),
 ('', 32),
 ('', 21),
 ('', 16),
 ('', 15),
 ('', 15),
 ('', 14),
 ('', 14),
 ('', 13),
 ('', 13),
 ('', 13),
 ('', 13),
 ('', 12),
 ('', 11),
 ('', 11),
 ('', 11),
 ('', 10)]
```

```
emoji_summary['top_emoji_text'][:20]
```

```
emoji_summary['top_emoji_groups']
```

```
[('Smileys & Emotion', 601),
('Food & Drink', 210),
('People & Body', 97),
('Symbols', 75),
('Travel & Places', 67),
('Animals & Nature', 33),
('Objects', 29),
('Activities', 26),
('Flags', 11)]
```

```
emoji_summary['top_emoji_sub_groups']
```

extract(*text_list*, *regex*, *key_name*, *extracted=None*, ***kwargs*)

Return a summary dictionary about arbitrary matches in `text_list`.

This function is used by other specialized functions to extract certain elements (hashtags, mentions, emojis, etc.). It can be used for other arbitrary elements/matches. You only need to provide your own regex.

Parameters

- **text_list** (*list*) -- Any list of strings (social posts, page titles, etc.)
- **regex** (*str*) -- The regex pattern to use for extraction.
- **key_name** (*str*) -- The name of the object extracted in singular form (hashtag, mention, etc.)
- **extracted** (*list(list)*) -- List of lists, optional. If the regex is not straightforward, and matches need to be made with special code, provide the extracted words/matches as a list for each element of `text_list`.
- **kwargs** (*mapping*) -- Other kwargs that might be needed.

Return summary A dictionary summarizing the extracted data.

extract_currency(*text_list*, *left_chars=20*, *right_chars=20*)

Return a summary dictionary about currency symbols in `text_list`

Get a summary of the number of currency symbols, their frequency, the top ones, and more.

Parameters

- **text_list** (*list*) -- A list of text strings.
- **left_chars** (*int*) -- The number of characters to extract, to the left of the symbol when getting `surrounding_text`
- **right_chars** (*int*) -- The number of characters to extract, to the left of the symbol when getting `surrounding_text`

Returns summary A dictionary with various stats about currencies

```
>>> posts = ['today 1 is around $4k', 'and  in £ & €?', 'no idea']
>>> currency_summary = extract_currency(posts)
>>> currency_summary.keys()
```

(continues on next page)

(continued from previous page)

```
dict_keys(['currency_symbols', 'currency_symbols_flat',
'currency_symbol_counts', 'currency_symbol_freq',
'top_currency_symbols', 'overview', 'currency_symbol_names'])
```

```
>>> currency_summary['currency_symbols']
[['', '$'], ['', '£', '€'], []]
```

A simple extract of currencies from each of the posts. An empty list if none exist

```
>>> currency_summary['currency_symbols_flat']
['', '$', '', '£', '€']
```

All currency symbols in one flat list.

```
>>> currency_summary['currency_symbol_counts']
[2, 3, 0]
```

The count of currency symbols per post.

```
>>> currency_summary['currency_symbol_freq']
[(0, 1), (2, 1), (3, 1)]
```

Shows how many posts had 0, 1, 2, 3, etc. currency symbols (number_of_symbols, count)

```
>>> currency_summary['top_currency_symbols']
[('', 2), ('$', 1), ('£', 1), ('€', 1)]
```

```
>>> currency_summary['currency_symbol_names']
[['bitcoin sign', 'dollar sign'], ['bitcoin sign', 'pound sign',
'euro sign'], []]
```

```
>>> currency_summary['surrounding_text']
[['today 1 is around $4k'], ['and  in £ & €?'], []]
```

```
>>> extract_currency(posts, 5, 5)['surrounding_text']
[['oday 1 is ', 'ound $4k'], ['and  in £', ' & €?'], []]
```

```
>>> extract_currency(posts, 0, 3)['surrounding_text']
[['1 i', '$4k'], [' in', '£ & ', '€?'], []]
```

```
>>> currency_summary['overview']
{'num_posts': 3,
'num_currency_symbols': 5,
'currency_symbols_per_post': 1.6666666666666667,
'unique_currency_symbols': 4}
```

extract_exclamations(*text_list*)

Return a summary dictionary about exclamation (mark)s in `text_list`

Get a summary of the number of exclamation marks, their frequency, the top ones, as well the exclamations written/said.

Parameters **text_list** (*list*) -- A list of text strings.

Returns summary A dictionary with various stats about exclamations

```
>>> posts = ['Who are you!', 'What is this!', 'No exclamation here?']
>>> exclamation_summary = extract_exclamations(posts)
>>> exclamation_summary.keys()
dict_keys(['exclamation_marks', 'exclamation_marks_flat',
'exclamation_mark_counts', 'exclamation_mark_freq',
'top_exclamation_marks', 'overview', 'exclamation_mark_names',
'exclamation_text'])
```

```
>>> exclamation_summary['exclamation_marks']
[['!'], ['!'], []]
```

A simple extract of exclamation marks from each of the posts. An empty list if none exist

```
>>> exclamation_summary['exclamation_marks_flat']
['!', '!']
```

All exclamation marks in one flat list.

```
>>> exclamation_summary['exclamation_mark_counts']
[1, 1, 0]
```

The count of exclamation marks per post.

```
>>> exclamation_summary['exclamation_mark_freq']
[(0, 1), (1, 2)]
```

Shows how many posts had 0, 1, 2, 3, etc. exclamation marks (number_of_symbols, count)

```
>>> exclamation_summary['top_exclamation_marks']
[('!', 2)]
```

Might be interesting if you have different types of exclamation marks

```
>>> exclamation_summary['exclamation_mark_names']
[['exclamation mark'], ['exclamation mark'], []]
```

```
>>> exclamation_summary['overview']
{'num_posts': 3,
'num_exclamation_marks': 2,
'exclamation_marks_per_post': 0.6666666666666666,
'unique_exclamation_marks': 1}
```

```
>>> posts2 = ["don't go there!", '.  !', '¡Hola! ¿cómo estás?',
... 'a few different exclamation marks! make sure you see them!']
```

```
>>> exclamation_summary = extract_exclamations(posts2)
```

```
>>> exclamation_summary['exclamation_marks']
[['!'], ['!'], ['¡', '!'], ['!', '!']]
# might be displayed in opposite order due to RTL exclamation mark
A simple extract of exclamation marks from each of the posts.
An empty list if none exist
```

```
>>> exclamation_summary['exclamation_marks_flat']
['!', '!', '¡', '!', '!', '!']
```

All exclamation marks in one flat list.

```
>>> exclamation_summary['exclamation_mark_counts']
[1, 1, 2, 2]
```

The count of exclamation marks per post.

```
>>> exclamation_summary['exclamation_mark_freq']
[(1, 2), (2, 2)]
```

Shows how many posts had 0, 1, 2, 3, etc. exclamation marks (number_of_symbols, count)

```
>>> exclamation_summary['top_exclamation_marks']
[('!', 5), ('¡', 1)]
```

Might be interesting if you have different types of exclamation marks

```
>>> exclamation_summary['exclamation_mark_names']
[['exclamation mark'], ['exclamation mark'],
['inverted exclamation mark', 'exclamation mark'],
['exclamation mark', 'exclamation mark']]
```

```
>>> exclamation_summary['overview']
{'num_posts': 4,
'num_exclamation_marks': 6,
'exclamation_marks_per_post': 1.5,
'unique_exclamation_marks': 4}
```

extract_hashtags(*text_list*)

Return a summary dictionary about hashtags in `text_list`

Get a summary of the number of hashtags, their frequency, the top ones, and more.

Parameters **text_list** (*list*) -- A list of text strings.

Returns **summary** A dictionary with various stats about hashtags

```
>>> posts = ['i like #blue', 'i like #green and #blue', 'i like all']
>>> hashtag_summary = extract_hashtags(posts)
>>> hashtag_summary.keys()
dict_keys(['hashtags', 'hashtags_flat', 'hashtag_counts', 'hashtag_freq',
'top_hashtags', 'overview'])
```

```
>>> hashtag_summary['hashtags']
[['#blue'], ['#green', '#blue'], []]
```

A simple extract of hashtags from each of the posts. An empty list if none exist

```
>>> hashtag_summary['hashtags_flat']
['#blue', '#green', '#blue']
```

All hashtags in one flat list.

```
>>> hashtag_summary['hashtag_counts']
[1, 2, 0]
```

The count of hashtags per post.

```
>>> hashtag_summary['hashtag_freq']
[(0, 1), (1, 1), (2, 1)]
```

Shows how many posts had 0, 1, 2, 3, etc. hashtags (number_of_hashtags, count)

```
>>> hashtag_summary['top_hashtags']
[('#blue', 2), ('#green', 1)]
```

```
>>> hashtag_summary['overview']
{'num_posts': 3,
 'num_hashtags': 3,
 'hashtags_per_post': 1.0,
 'unique_hashtags': 2}
```

extract_intense_words(*text_list*, *min_reps=3*)

Return a summary dictionary about intense words in `text_list`

Get all instances of usage of intense words (positive or negative), using words that have `min_reps` or more repetitions of characters. "I looooooveeee youuuuuuu", and "I haaattteeee youuuuuu" are both intense.

Parameters

- **text_list** (*list*) -- A text list from which to extract intense words
- **min_reps** (*int*) -- The number of times a character has to be repeated for the word to be considered intense.

Returns summary A dictionary with various stats about intense words

extract_mentions(*text_list*)

Return a summary dictionary about mentions in `text_list`

Get a summary of the number of mentions, their frequency, the top ones, and more.

Parameters **text_list** (*list*) -- A list of text strings.

Returns summary A dictionary with various stats about mentions

```
>>> posts = ['hello @john and @jenny', 'hi there @john', 'good morning']
>>> mention_summary = extract_mentions(posts)
>>> mention_summary.keys()
dict_keys(['mentions', 'mentions_flat', 'mention_counts', 'mention_freq',
'top_mentions', 'overview'])
```

```
>>> mention_summary['mentions']
[['@john', '@jenny'], ['@john'], []]
```

A simple extract of mentions from each of the posts. An empty list if none exist

```
>>> mention_summary['mentions_flat']
['@john', '@jenny', '@john']
```

All mentions in one flat list.

```
>>> mention_summary['mention_counts']
[2, 1, 0]
```

The count of mentions per post.

```
>>> mention_summary['mention_freq']
[(0, 1), (1, 1), (2, 1)]
```

Shows how many posts had 0, 1, 2, 3, etc. mentions (number_of_mentions, count)

```
>>> mention_summary['top_mentions']
[('@john', 2), ('@jenny', 1)]
```

```
>>> mention_summary['overview']
{'num_posts': 3, # number of posts
 'num_mentions': 3,
 'mentions_per_post': 1.0,
 'unique_mentions': 2}
```

extract_numbers(*text_list*, *number_separators=('.', ',', '-')*)

Return a summary dictionary about numbers in `text_list`, separated by any of `number_separators`

Get a summary of the number of numbers, their frequency, the top ones, and more. Typically, numbers would contain separators to make them easier to read, so these are included by default, which you can modify.

Parameters

- **text_list** (*list*) -- A list of text strings.
- **number_separators** (*list(str)*) -- A list of separators that you want to be included as part of the extracted numbers.

Returns summary A dictionary with various stats about the numbers

```
>>> posts = ['text before 123', '123,456 text after', 'phone 333-444-555',
'multiple 123,456 and 123.456.789']
>>> number_summary = extract_numbers(posts)
>>> number_summary.keys()
dict_keys(['numbers', 'numbers_flat', 'number_counts', 'number_freq',
'top_numbers', 'overview'])
```

```
>>> number_summary['numbers']
[['123'], ['123,456'], ['333-444-555'], ['123,456', '123.456.789']]
```

A simple extract of number from each of the posts. An empty list if none exist

```
>>> number_summary['numbers_flat']
['123', '123,456', '333-444-555', '123,456', '123.456.789']
```

All numbers in one flat list.

```
>>> number_summary['number_counts']
[1, 1, 1, 2]
```

The count of numbers per post.

```
>>> number_summary['number_freq']
[(1, 3), (2, 1)]
```

Shows how many posts had 0, 1, 2, 3, etc. numbers (number_of_numbers, count)

```
>>> number_summary['top_numbers']
[('123,456', 2), ('123', 1), ('333-444-555', 1), ('123.456.789', 1)]
```

```
>>> number_summary['overview']
{'num_posts': 4,
 'num_numbers': 5,
 'numbers_per_post': 1.25,
 'unique_numbers': 4}
```

extract_questions(*text_list*)

Return a summary dictionary about question(mark)s in `text_list`

Get a summary of the number of question marks, their frequency, the top ones, as well the questions asked.

Parameters **text_list** (*list*) -- A list of text strings.

Returns **summary** A dictionary with various stats about questions

```
>>> posts = ['How are you?', 'What is this?', 'No question Here!']
>>> question_summary = extract_questions(posts)
>>> question_summary.keys()
dict_keys(['question_marks', 'question_marks_flat',
'question_mark_counts', 'question_mark_freq', 'top_question_marks',
'overview', 'question_mark_names', 'question_text'])
```

```
>>> question_summary['question_marks']
[['?'], ['?'], []]
```

A simple extract of question marks from each of the posts. An empty list if none exist

```
>>> question_summary['question_marks_flat']
['?', '?']
```

All question marks in one flat list.

```
>>> question_summary['question_mark_counts']
[1, 1, 0]
```

The count of question marks per post.

```
>>> question_summary['question_mark_freq']
[(0, 1), (1, 2)]
```

Shows how many posts had 0, 1, 2, 3, etc. question marks (number_of_symbols, count)

```
>>> question_summary['top_question_marks']
[('?', 2)]
```

Might be interesting if you have different types of question marks (Arabic, Spanish, Greek, Armenian, or other)

```
>>> question_summary['question_mark_names']
[['question mark'], ['question mark'], []]
```

```
>>> question_summary['overview']
{'num_posts': 3,
'num_question_marks': 2,
'question_marks_per_post': 0.6666666666666666,
'unique_question_marks': 1}
```

```
>>> posts2 = [' ', '.  ', 'Hola, ¿cómo estás?',
... 'Can you see the new questions? Did you notice the different marks?']
```

```
>>> question_summary = extract_questions(posts2)
```

```
>>> question_summary['question_marks']
[[''], [''], ['¿', '?'], ['?', '?']]
# might be displayed in opposite order due to RTL question mark
A simple extract of question marks from each of the posts. An empty list if
none exist
```

```
>>> question_summary['question_marks_flat']
['', '', '¿', '?', '?', '?']
```

All question marks in one flat list.

```
>>> question_summary['question_mark_counts']
[1, 1, 2, 2]
```

The count of question marks per post.

```
>>> question_summary['question_mark_freq']
[(1, 2), (2, 2)]
```

Shows how many posts had 0, 1, 2, 3, etc. question marks (number_of_symbols, count)

```
>>> question_summary['top_question_marks']
[('?', 3), ('', 1), ('', 1), ('¿', 1)]
```

Might be interesting if you have different types of question marks (Arabic, Spanish, Greek, Armenian, or other)

```
>>> question_summary['question_mark_names']
[['greek question mark'], ['arabic question mark'],
['inverted question mark', 'question mark'],
['question mark', 'question mark']]
# correct order
```

```
>>> question_summary['overview']
{'num_posts': 4,
'num_question_marks': 6,
'question_marks_per_post': 1.5,
'unique_question_marks': 4}
```

extract_urls(*text_list*)

Return a summary dictionary about URLs in `text_list`

Get a summary of the number of URLs, their frequency, the top ones, and more. This does NOT validate URLs, www.a.b would count as a URL

Parameters **text_list** (*list*) -- A list of text strings.

Returns summary A dictionary with various stats about URLs

```
>>> posts = ['one link http://example.com', 'two: http://a.com www.b.com',
...          'no links here',
...          'long url http://example.com/one/two/?1=one&2=two']
>>> url_summary = extract_urls(posts)
>>> url_summary.keys()
dict_keys(['urls', 'urls_flat', 'url_counts', 'url_freq',
'top_urls', 'overview', 'top_domains', 'top_tlds'])
```

```
>>> url_summary['urls']
[['http://example.com'],
 ['http://a.com', 'http://www.b.com'],
 [],
 ['http://example.com/one/two/?1=one&2=two']]
```

A simple extract of urls from each of the posts. An empty list if none exist

```
>>> url_summary['urls_flat']
['http://example.com', 'http://a.com', 'http://www.b.com',
 'http://example.com/one/two/?1=one&2=two']
```

All urls in one flat list.

```
>>> url_summary['url_counts']
[1, 2, 0, 1]
```

The count of urls per post.

```
>>> url_summary['url_freq']
[(0, 1), (1, 2), (2, 1)]
```

Shows how many posts had 0, 1, 2, 3, etc. urls (number_of_urls, count)

```
>>> url_summary['top_urls']
[('http://example.com', 1), ('http://a.com', 1), ('http://www.b.com', 1),
 ('http://example.com/one/two/?1=one&2=two', 1)]
```

```
>>> url_summary['top_domains']
[('example.com', 2), ('a.com', 1), ('www.b.com', 1)]
```

```
>>> url_summary['top_tlds']
[('com', 4)]
```

```
>>> url_summary['overview']
{'num_posts': 4,
```

(continues on next page)

(continued from previous page)

```
 'num_urls': 4,
 'urls_per_post': 1.0,
 'unique_urls': 4}
```

extract_words(*text_list*, *words_to_extract*, *entire_words_only=False*)

Return a summary dictionary about `words_to_extract` in `text_list`.

Get a summary of the number of words, their frequency, the top ones, and more.

Parameters

- **text_list** (*list*) -- A list of text strings.
- **words_to_extract** (*list*) -- A list of words to extract from `text_list`.
- **entire_words_only** (*bool*) -- Whether or not to find only complete words (as specified by `words_to_find`) or find any any of the words as part of longer strings.

Returns summary A dictionary with various stats about the words

```
>>> posts = ['there is rain, it is raining', 'there is snow and rain',
             'there is no rain, it is snowing', 'there is nothing']
>>> word_summary = extract_words(posts, ['rain', 'snow'], True)
>>> word_summary.keys()
dict_keys(['words', 'words_flat', 'word_counts', 'word_freq',
'top_words', 'overview'])
```

```
>>> word_summary['overview']
{'num_posts': 4,
 'num_words': 4,
 'words_per_post': 1,
 'unique_words': 2}
```

```
>>> word_summary['words']
[['rain'], ['snow', 'rain'], ['rain'], []]
```

A simple extract of mentions from each of the posts. An empty list if none exist

```
>>> word_summary['words_flat']
['rain', 'snow', 'rain', 'rain']
```

All mentions in one flat list.

```
>>> word_summary['word_counts']
[1, 2, 1, 0]
```

The count of mentions for each post.

```
>>> word_summary['word_freq']
[(0, 1) (1, 2), (2, 1)]
```

Shows how many posts had 0, 1, 2, 3, etc. words (number_of_words, count)

```
>>> word_summary['top_words']
[('rain', 3), ('snow', 1)]
```

Check the same posts extracting any occurrence of the specified words with `entire_words_only=False`:

```
>>> word_summary = extract_words(posts, ['rain', 'snow'], False)
```

```
>>> word_summary['overview']
{'num_posts': 4, # number of posts
 'num_words': 6,
 'words_per_post': 1.5,
 'unique_words': 4}
```

```
>>> word_summary['words']
[['rain', 'raining'], ['snow', 'rain'], ['rain', 'snowing'], []]
```

Note that the extracted words are the complete words so you can see where they occurred. In case "training" was mentioned, you would see that it is not related to rain for example.

```
>>> word_summary['words_flat']
['rain', 'raining', 'snow', 'rain', 'rain', 'snowing']
```

All mentions in one flat list.

```
>>> word_summary['word_counts']
[2, 2, 2, 0]
```

```
>>> word_summary['word_freq']
[(0, 1), (2, 3)]
```

Shows how many posts had 0, 1, 2, 3, etc. words (number_of_words, count)

```
>>> word_summary['top_words']
[('rain', 3), ('raining', 1), ('snow', 1), ('snowing', 1)]
```

1.19 Stopwords in Several Languages

List of stopwords by the spaCy[1] package, useful in text mining, analyzing content of social media posts, tweets, web pages, keywords, etc.

Each list is accessible as part of a dictionary `stopwords` which is a normal Python dictionary.

1.19.1 Stopword Languages

- Arabic
- Azerbaijani
- Bengali
- Catalan
- Chinese
- Croatian

[1] Copyright (C) 2016 ExplosionAI UG (haftungsbeschränkt), 2016 spaCy GmbH, 2015 Matthew Honnibal

- Danish
- Dutch
- English
- Finnish
- French
- German
- Greek
- Hebrew
- Hindi
- Hungarian
- Indonesian
- Irish
- Italian
- Japanese
- Kazakh
- Nepali
- Norwegian
- Persian
- Polish
- Portuguese
- Romanian
- Russian
- Sinhala
- Spanish
- Swedish
- Tagalog
- Tamil
- Tatar
- Telugu
- Thai
- Turkish
- Ukrainian
- Urdu
- Vietnamese

You can easily explore the available languages and get (and optionally modify) the stopwords by accessing the dictionary as follows:

```
import advertools as adv
adv.stopwords.keys()
```

```
dict_keys(['arabic', 'azerbaijani', 'bengali', 'catalan', 'chinese',
'croatian', 'danish', 'dutch', 'english', 'finnish', 'french',
'german', 'greek', 'hebrew', 'hindi', 'hungarian', 'indonesian',
'irish', 'italian', 'japanese', 'kazakh', 'nepali', 'norwegian',
'persian', 'polish', 'portuguese', 'romanian', 'russian', 'sinhala',
'spanish', 'swedish', 'tagalog', 'tamil', 'tatar', 'telugu', 'thai',
'turkish', 'ukrainian', 'urdu', 'vietnamese'])
```

You can also access the stopwords of a certain language:

```
print(sorted(adv.stopwords['english'])[:5])

print(sorted(adv.stopwords['german'])[:5])
```

1.20 Text Analysis

1.20.1 Absolute and Weighted Word Count

When analyzing a corpus of documents (I'll simply call it a text list), one of the main tasks to accomplish to start text mining is to first count the words. While there are many text mining techniques and approaches, the *word_frequency()* function works mainly by counting words in a text list. A "word" is defined as a sequence of characters split by whitespace(s), and stripped of non-word characters (commas, dots, quotation marks, etc.). A "word" is actually a phrase consisting of one word, but you have the option of getting phrases that have two words, or more. This can be done simply by providing a value for the `phrase_len` parameter.

Absolute vs Weighted Frequency

In social media reports, analytics, keyword reports, url and page reports, we get more information than simply the text. We get numbers describing those posts or page titles, or product names, or whatever the text list might contain. Numbers can be pageviews, shares, likes, retweets, sales, bounces, sales, etc. Since we have numbers to quantify those phrases, we can improve our counting by taking into consideration the number list that comes with the text list.

For example, if you have an e-commerce site that has two products, let's say you have bags and shoes, then your products are split 50:50 between bags and shoes. But what if you learn that shoes generate 80% of your sales? Although shoes form half your products, they generate 80% of your revenue. So the *weighted count* of your products is 80:20.

Let's say two people post two different posts on a social media platform. One of them says, "It's raining", and the other says, "It's snowing". As in the above example, the content is split 50:50 between "raining" and "snowing", but we get a much more informative picture if we get the number of followers of each of those accounts (or the number of shares, likes, etc.). If one of them has a thousand followers, and other has a million (which is typical on social media, as well as in pageviews report, e-commerce and most other datasets), then you get a completely different picture about your dataset.

These two simple examples contain two posts, and a word each. The *word_frequency()* function can provide insight on hidden trends especially in large datasets, and when the sentences or phrases are also longer then a word or two each.

Let's take a look at how to use the *word_frequency()* function, and what the available parameters and options are.

text_list The list of phrases or documents that you want to analyze. Here are some possible ideas that you might use this for:

- keywords, whether in a PPC or SEO report
- page titles in an analytics report
- social media posts (tweets, Facebook posts, YouTube video titles or descriptions etc.)
- e-commerce reports (where the text would be the product names)

num_list Ideally, if you have more than one column describing `text_list` you should experiment with different options. Try weighting the words by pageviews, then try by bounce rate and see if you get different interesting findings. With e-commerce reports, you can see which word appears the most, and which word is associated with more revenue.

phrase_len You should also experiment with different lengths of phrases. In many cases, one-word phrases might not be as meaningful as two-words or three.

regex The default is to simply split words by whitespace, and provide phrases of length `phrase_len`. But you may want to count the occurrences of certain patterns of text. Check out the *regex* module for the available regular expressions that might be interesting. Some of the pre-defined ones are hashtags, mentions, questions, emoji, currencies, and more.

rm_words A list of words to remove and ignore from the count. Known as stop-words these are the most frequently used words in a language, the most used, but don't add much meaning to the content (a, and, of, the, if, etc.). By default a set of English stopwords is provided (which you can check and possibly may want to modify), or run `adv.stopwords.keys()` to get a list of all the available stopwords in the available languages. In some cases (like page titles for example), you might get "words" that need to be removed as well, like the pipe "|" character for example.

extra_info The returned DataFrame contains the default columns `[word, abs_freq, wtd_freq, rel_value]`. You can get extra columns for percentages and cumulative percentages that add perspective to the other columns. Set this parameter to `True` if you want that.

Below are all the columns of the returned DataFrame:

`word`	Words in the document list each on its own row. The length of these words is determined by `phrase_len`, essentially phrases if containing more than one word each.
`abs_freq`	The number of occurrences of each word in all the documents.
`wtd_freq`	Every occurrence of `word` multiplied by its respective value in `num_list`.
`rel_value`	`wtd_freq` divided by `abs_freq`, showing the value per occurrence of `word`
`abs_perc`	Absolute frequency percentage.
`abs_perc_cum`	Cumulative absolute percentage.
`wtd_freq_perc`	Weighted frequency percentage.
`wtd_freq_perc_cum`	Cumulative weighted frequency percentage.

```
import advertools as adv
import pandas as pd
tweets = pd.read_csv('data/tweets.csv')
tweets
```

	tweet_text	fol-low-ers_count
0	@AERIALMAGZC @penguinnyyyyy you won't be afraid if I give you a real reason :D	157
1	Vibing in the office to #Metallica when the boss is on a coffee break #TheOffice https://t.co/U5vdYevvfe	4687
2	I feel like Ann says she likes coffee and then gets drinks that are 99% sugar and 1% coffee https://t.co/HfuBV4v3aY	104
3	A venti iced coffee with four pumps of white mocha, sweet cream and caramel drizzle might just be my new favorite drink. Shout out to TikTok lol	126
4	I was never a coffee person until I had kids. this cup is a life saver. https://t.co/Zo0CnVuiGj	1595
5	Who's excited about our next Coffee Chat? We know we are! We're also adding Representative John Bradford to this lineup to discuss redistricting in the area. You won't want to miss it! RSVP: https://t.co/R3YNJjJCUG Join the meeting: https://t.co/Ho4Kx7ZZ24 https://t.co/KfPdR3hupY	5004
6	he paid for my coffee= husband	165
7	It's nipply outside, and now I side too :) That sounds like blowjob in front of a fire and visit with coffee after :) I'm still out of coffee I could have green tea instead Hahahahahahaha I want to spend the morning pampering you ...	0
8	Good morning I hope everyone has a great Tuesday morning. Enjoy your day and coffee	189
9	@MarvinMilton2 I nearly choked on my coffee	1160

```
word_freq = adv.word_frequency(text_list=tweets['tweet_text'],
                               num_list=tweets['followers_count'])

# try sorting by 'abs_freq', 'wtd_freq', and 'rel_value':
word_freq.sort_values(by='abs_freq', ascending=False).head(25)
```

word_frequency(*text_list*, *num_list=None*, *phrase_len=1*, *regex=None*, *rm_words={'a', 'about', 'above', 'across', 'after', 'afterwards', 'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', 'am', 'among', 'amongst', 'amount', 'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere', 'are', 'around', 'as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being', 'below', 'beside', 'besides', 'between', 'beyond', 'both', 'bottom', 'but', 'by', 'ca', 'call', 'can', 'cannot', 'could', 'did', 'do', 'does', 'doing', 'done', 'down', 'due', 'during', 'each', 'eight', 'either', 'eleven', 'else', 'elsewhere', 'empty', 'enough', 'even', 'ever', 'every', 'everyone', 'everything', 'everywhere', 'except', 'few', 'fifteen', 'fifty', 'first', 'five', 'for', 'former', 'formerly', 'forty', 'four', 'from', 'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'have', 'he', 'hence', 'her', 'here', 'hereafter', 'hereby', 'herein', 'hereupon', 'hers', 'herself', 'him', 'himself', 'his', 'how', 'however', 'hundred', 'i', 'if', 'in', 'indeed', 'into', 'is', 'it', 'its', 'itself', 'just', 'keep', 'last', 'latter', 'latterly', 'least', 'less', 'made', 'make', 'many', 'may', 'me', 'meanwhile', 'might', 'mine', 'more', 'moreover', 'most', 'mostly', 'move', 'much', 'must', 'my', 'myself', 'name', 'namely', 'neither', 'never', 'nevertheless', 'next', 'nine', 'no', 'nobody', 'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere', 'of', 'off', 'often', 'on', 'once', 'one', 'only', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps', 'please', 'put', 'quite', 'rather', 're', 'really', 'regarding', 'same', 'say', 'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'take', 'ten', 'than', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'third', 'this', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty', 'two', 'under', 'unless', 'until', 'up', 'upon', 'us', 'used', 'using', 'various', 'very', 'via', 'was', 'we', 'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever', 'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yourselves'}*, *extra_info=False*)

Count the absolute as well as the weighted frequency of words in `text_list` (based on `num_list`).

Parameters

- **text_list** (*list*) -- Typically short phrases, but could be any list of full blown documents. Usually, you would use this to analyze tweets, book titles, URLs, etc.
- **num_list** (*list*) -- A list of numbers with the same length as `text_list`, describing a certain attribute of these 'documents'; views, retweets, sales, etc.
- **regex** (*str*) -- The regex used to split words. Doesn't need changing in most cases.
- **phrase_len** (*int*) -- the length in words of each token the text is split into, defaults to 1.
- **rm_words** (*set*) -- Words to remove from the list a.k.a 'stop-words'. The default uses. To get all available languages run `adv.stopwords.keys()`
- **extra_info** (*bool*) -- Whether or not to give additional metrics about the frequencies

Returns abs_wtd_df absolute and weighted DataFrame.

```
>>> text_list = ['apple orange', 'apple orange banana',
...              'apple kiwi', 'kiwi mango']
>>> num_list = [100, 100, 100, 400]
```

```
>>> adv.word_frequency(text_list, num_list)
     word  abs_freq  wtd_freq  rel_value
0    kiwi         2       500      250.0
1   mango         1       400      400.0
2   apple         3       300      100.0
```

(continues on next page)

(continued from previous page)

```
3  orange          2       200      100.0
4  banana          1       100      100.0
```

Although "kiwi" occurred twice `abs_freq`, and "apple" occurred three times, the phrases in which "kiwi" appear have a total score of 500, so it beats "apple" on `wtd_freq` even though "apple" wins on `abs_freq`. You can sort by any of the columns of course. `rel_value` shows the value per occurrence of each word, as you can see, it is simply obtained by dividing `wtd_freq` by `abs_freq`.

```
>>> adv.word_frequency(text_list)  # num_list values default to 1 each
     word  abs_freq  wtd_freq  rel_value
0   apple         3         3        1.0
1  orange         2         2        1.0
2    kiwi         2         2        1.0
3  banana         1         1        1.0
4   mango         1         1        1.0
```

```
>>> text_list2 = ['my favorite color is blue',
... 'my favorite color is green', 'the best color is green',
... 'i love the color black']
```

Setting `phrase_len` to 2, "words" become two-word phrases instead. Note that we are setting `rm_words` to the empty list so we can keep the stopwords and see if that makes sense:

```
>>> word_frequency(text_list2, phrase_len=2, rm_words=[])
             word  abs_freq  wtd_freq  rel_value
0         color is         3         3        1.0
1      my favorite         2         2        1.0
2   favorite color         2         2        1.0
3         is green         2         2        1.0
4          is blue         1         1        1.0
5         the best         1         1        1.0
6       best color         1         1        1.0
7           i love         1         1        1.0
8         love the         1         1        1.0
9        the color         1         1        1.0
10     color black         1         1        1.0
```

The same result as above showing all possible columns by setting `extra_info` to `True`:

```
>>> adv.word_frequency(text_list, num_list, extra_info=True)
     word  abs_freq  abs_perc  abs_perc_cum  wtd_freq  wtd_freq_perc  wtd_freq_perc_
↪cum  rel_value
0    kiwi         2  0.222222      0.222222       500       0.333333           0.
↪333333      250.0
1   mango         1  0.111111      0.333333       400       0.266667           0.
↪600000      400.0
2   apple         3  0.333333      0.666667       300       0.200000           0.
↪800000      100.0
3  orange         2  0.222222      0.888889       200       0.133333           0.
↪933333      100.0
4  banana         1  0.111111      1.000000       100       0.066667           1.
↪000000      100.0
```

1.21 Tokenize Words (N-grams)

As word counting is an essential step in any text mining task, you first have to split the text into words.

The *word_tokenize()* function achieves that by splitting the text by whitespace. Another important thing it does after splitting is to trim the words of any non-word characters (commas, dots, exclamation marks, etc.).

You also have the option of specifying the length of the words that you want. The default is 2, which can be set through the `phrase_len` parameter.

This function is mainly a helper function for *word_frequency* to help with counting words and/or phrases.

word_tokenize(*text_list*, *phrase_len=2*)

Split `text_list` into phrases of length `phrase_len` words each.

A "word" is any string between white spaces (or beginning or end of string) with delimiters stripped from both sides. Delimiters include quotes, question marks, parentheses, etc. Any delimiter contained within the string remains. See examples below.

Parameters

- **text_list** -- List of strings.
- **phrase_len** -- Length of word tokens, defaults to 2.

Return tokenized List of lists, split according to `phrase_len`.

```
>>> t = ['split me into length-n-words',
... 'commas, (parentheses) get removed!',
... 'commas within text remain $1,000, but not the trailing commas.']
```

```
>>> word_tokenize(t, 1)
[['split', 'me', 'into', 'length-n-words'],
['commas', 'parentheses', 'get', 'removed'],
['commas', 'within', 'text', 'remain', '$1,000',
'but', 'not', 'the', 'trailing', 'commas']]
```

The comma inside "$1,000" as well as the dollar sign remain, as they are part of the "word", but the trailing comma is stripped.

```
>>> word_tokenize(t, 2)
[['split me', 'me into', 'into length-n-words'],
['commas parentheses', 'parentheses get', 'get removed'],
['commas within', 'within text', 'text remain', 'remain $1,000',
'$1,000 but', 'but not', 'not the', 'the trailing', 'trailing commas']]
```

```
>>> word_tokenize(t, 3)
[['split me into', 'me into length-n-words'],
['commas parentheses get', 'parentheses get removed'],
['commas within text', 'within text remain', 'text remain $1,000',
'remain $1,000 but', '$1,000 but not', 'but not the',
'not the trailing', 'the trailing commas']]
```

1.22 Twitter Data API

Easily connect to the Twitter API and start your analysis immediately.

Main Features:

1 **Get the results in a DataFrame**: With the exception of three functions that return a list of ID's, everything else returns a pandas DataFrame, ready to use. This allows you to spend more time analyzing data, and less time figuring out the structure of the JSON response object. It's not complicated or anything, just takes time.

2 **Manage looping and merging**: there is a limit on how many results you get per request (typically in the 100 - 200 range), several requests have to be made, and merged together. Not all responses have the same structure, so this is also handled. You only have to provide the number of responses you want through the `count` parameter where applicable (provided you are within your app's rate limits).

3 **Unnesting nested objects**: Many response objects contain very rich embedded data, which is usually meta data about the response. For example, when you request tweets, you get a user object along with that. This is very helpful in better understanding who made the tweet, and how influential/credible they are.

4 **Documentation**: All available parameters are included in the function signatures, to make it easier to explore interactively, as well as descriptions of the parameters imported from the Twitter documentation.

1.22.1 Authentication

Before starting you will have to create an app through developer.twitter.com, and then you can get your authentication keys from your dashboard. Then you can authenticate as follows:

```
>>> auth_params = {
...     'app_key': 'YOUR_APP_KEY',
...     'app_secret': 'YOUR_APP_SECRET',
...     'oauth_token': 'YOUR_OAUTH_TOKEN',
...     'oauth_token_secret': 'YOUR_OAUTH_TOKEN_SECRET',
... }
>>> import advertools as adv
>>> adv.twitter.set_auth_params(**auth_params)
```

Now every request you send will include your `auth_params` in it, and if valid you will get the respective response, for example:

```
>>> python_tweets = adv.twitter.search(q='#python', tweet_mode='extended')
```

Make sure you always specify `tweet_mode='extended'` because otherwise you will get tweets that are 140 characters long.

When you have tweets and user data in the DataFrame, the column names would be prepended with `tweet_` and `user_` to make it clear what the data belong to.

1.22.2 Functions

authenticate(*func*)

Used internally, please use set_auth_params for authentication.

get_application_rate_limit_status(*consumed_only=True*)

Returns the current rate limits for methods belonging to the specified resource families.

Parameters **consumed_only** -- Whether or not to return only items that have been consumed. Otherwise returns the full list.

https://developer.twitter.com/en/docs/developer-utilities/rate-limit-status/api-reference/get-application-rate_limit_status

get_available_trends()

Returns the locations that Twitter has trending topic information for.

https://developer.twitter.com/en/docs/trends/locations-with-trending-topics/api-reference/get-trends-available

get_favorites(*user_id=None*, *screen_name=None*, *count=None*, *since_id=None*, *max_id=None*, *include_entities=None*, *tweet_mode=None*)

Returns the 20 most recent Tweets favorited by the authenticating or specified user.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **since_id** -- (int - optional) Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
- **max_id** -- (int - optional) Returns results with an ID less than (that is, older than) or equal to the specified ID.
- **include_entities** -- (bool - optional) The entities node will be omitted when set to False .
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-favorites-list

get_followers_ids(*user_id=None*, *screen_name=None*, *cursor=None*, *stringify_ids=None*, *count=None*)

Returns a cursored collection of user IDs for every user following the specified user.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results.

- **cursor** -- (cursor - semi-optional) Causes the list of connections to be broken into pages of no more than 5000 IDs at a time. The number of IDs returned is not guaranteed to be 5000 as suspended users are filtered out after connections are queried. If no cursor is provided, a value of -1 will be assumed, which is the first "page." The response from the API will include a previous_cursor and next_cursor to allow paging back and forth. See Using cursors to navigate collections for more information.
- **stringify_ids** -- (bool - optional) Some programming environments will not consume Twitter IDs due to their size. Provide this option to have IDs returned as strings instead. More about Twitter IDs.
- **count** -- (int - optional) Specifies the number of results to retrieve.

https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-ids

get_followers_list(*user_id=None*, *screen_name=None*, *cursor=None*, *count=None*, *skip_status=None*, *include_user_entities=None*)

Returns a cursored collection of user objects for users following the specified user.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results.
- **cursor** -- (cursor - semi-optional) Causes the results to be broken into pages. If no cursor is provided, a value of -1 will be assumed, which is the first "page." The response from the API will include a previous_cursor and next_cursor to allow paging back and forth. See Using cursors to navigate collections for more information.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **skip_status** -- (bool - optional) When set to True, statuses will not be included in the returned user objects. If set to any other value, statuses will be included.
- **include_user_entities** -- (bool - optional) The user object entities node will not be included when set to False.

https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-list

get_friends_ids(*user_id=None*, *screen_name=None*, *cursor=None*, *stringify_ids=None*, *count=None*)

Returns a cursored collection of user IDs for every user the specified user is following (otherwise known as their "friends").

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results.
- **cursor** -- (cursor - semi-optional) Causes the list of connections to be broken into pages of no more than 5000 IDs at a time. The number of IDs returned is not guaranteed to be 5000 as suspended users are filtered out after connections are queried. If no cursor is provided, a value of -1 will be assumed, which is the first "page." The response from the API will include a previous_cursor and next_cursor to allow paging back and forth. See Using cursors to navigate collections for more information.

- **stringify_ids** -- (bool - optional) Some programming environments will not consume Twitter IDs due to their size. Provide this option to have IDs returned as strings instead. More about Twitter IDs.
- **count** -- (int - optional) Specifies the number of results to retrieve.

https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-ids

get_friends_list(*user_id=None*, *screen_name=None*, *cursor=None*, *count=None*, *skip_status=None*, *include_user_entities=None*)

Returns a cursored collection of user objects for every user the specified user is following (otherwise known as their "friends").

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results.
- **cursor** -- (cursor - semi-optional) Causes the results to be broken into pages. If no cursor is provided, a value of -1 will be assumed, which is the first "page." The response from the API will include a previous_cursor and next_cursor to allow paging back and forth. See Using cursors to navigate collections for more information.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **skip_status** -- (bool - optional) When set to True statuses will not be included in the returned user objects.
- **include_user_entities** -- (bool - optional) The user object entities node will not be included when set to False.

https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-list

get_home_timeline(*count=None*, *since_id=None*, *max_id=None*, *trim_user=None*, *exclude_replies=None*, *include_entities=None*, *tweet_mode=None*)

Returns a collection of the most recent Tweets and retweets posted by the authenticating user and the users they follow.

Parameters

- **count** -- (int - optional) Specifies the number of results to retrieve.
- **since_id** -- (int - optional) Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
- **max_id** -- (int - optional) Returns results with an ID less than (that is, older than) or equal to the specified ID.
- **trim_user** -- (bool - optional) When set to True, each Tweet returned in a timeline will include a user object including only the status authors numerical ID. Omit this parameter to receive the complete user object.
- **exclude_replies** -- (bool - optional) This parameter will prevent replies from appearing in the returned timeline. Using exclude_replies with the count parameter will mean you will

receive up-to count Tweets — this is because the count parameter retrieves that many Tweets before filtering out retweets and replies.

- **include_entities** -- (bool - optional) The entities node will not be included when set to False.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-home_timeline

get_list_members(*list_id=None*, *slug=None*, *owner_screen_name=None*, *owner_id=None*, *count=None*, *cursor=None*, *include_entities=None*, *skip_status=None*)

Returns the members of the specified list.

Parameters

- **list_id** -- (str - required) The numerical id of the list.
- **slug** -- (str - required) You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you'll also have to specify the list owner using the owner_id or owner_screen_name parameters.
- **owner_screen_name** -- (str - optional) The screen name of the user who owns the list being requested by a slug.
- **owner_id** -- (int - optional) The user ID of the user who owns the list being requested by a slug.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **cursor** -- (cursor - semi-optional) Causes the collection of list members to be broken into "pages" of consistent sizes (specified by the count parameter). If no cursor is provided, a value of -1 will be assumed, which is the first "page." The response from the API will include a previous_cursor and next_cursor to allow paging back and forth. See Using cursors to navigate collections for more information.
- **include_entities** -- (bool - optional) The entities node will not be included when set to False.
- **skip_status** -- (bool - optional) When set to True statuses will not be included in the returned user objects.

https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/api-reference/get-lists-members

get_list_memberships(*user_id=None*, *screen_name=None*, *count=None*, *cursor=None*, *filter_to_owned_lists=None*)

Returns the lists the specified user has been added to.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results. Helpful for disambiguating when a valid user ID is also a valid screen name.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results. Helpful for disambiguating when a valid screen name is also a user ID.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **cursor** -- (cursor - optional) Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned in the response body's next_cursor and previous_cursor

attributes to page back and forth in the list. It is recommended to always use cursors when the method supports them. See Cursoring for more information.

- **filter_to_owned_lists** -- (bool - optional) When True, will return just lists the authenticating user owns, and the user represented by user_id or screen_name is a member of.

https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/api-reference/get-lists-memberships

get_list_statuses(*list_id=None*, *slug=None*, *owner_screen_name=None*, *owner_id=None*, *since_id=None*, *max_id=None*, *count=None*, *include_entities=None*, *include_rts=None*, *tweet_mode=None*)

Returns a timeline of tweets authored by members of the specified list.

Parameters

- **list_id** -- (str - required) The numerical id of the list.
- **slug** -- (str - required) You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you'll also have to specify the list owner using the owner_id or owner_screen_name parameters.
- **owner_screen_name** -- (str - optional) The screen name of the user who owns the list being requested by a slug .
- **owner_id** -- (int - optional) The user ID of the user who owns the list being requested by a slug .
- **since_id** -- (int - optional) Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
- **max_id** -- (int - optional) Returns results with an ID less than (that is, older than) or equal to the specified ID.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **include_entities** -- (bool - optional) Entities are ON by default in API 1.1, each tweet includes a node called "entities". This node offers a variety of metadata about the tweet in a discreet structure, including: user_mentions, urls, and hashtags. You can omit entities from the result by using include_entities=False
- **include_rts** -- (bool - optional) When set to True, the list timeline will contain native retweets (if they exist) in addition to the standard stream of tweets. The output format of retweeted tweets is identical to the representation you see in home_timeline.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/api-reference/get-lists-statuses

get_list_subscribers(*list_id=None*, *slug=None*, *owner_screen_name=None*, *owner_id=None*, *count=None*, *cursor=None*, *include_entities=None*, *skip_status=None*)

Returns the subscribers of the specified list.

Parameters

- **list_id** -- (str - required) The numerical id of the list.
- **slug** -- (str - required) You can identify a list by its slug instead of its numerical id. If you decide to do so, note that you'll also have to specify the list owner using the owner_id or owner_screen_name parameters.

- **owner_screen_name** -- (str - optional) The screen name of the user who owns the list being requested by a slug .
- **owner_id** -- (int - optional) The user ID of the user who owns the list being requested by a slug .
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **cursor** -- (cursor - optional) Breaks the results into pages. A single page contains 20 lists. Provide a value of -1 to begin paging. Provide values as returned in the response body's next_cursor and previous_cursor attributes to page back and forth in the list. See Using cursors to navigate collections for more information.
- **include_entities** -- (bool - optional) When set to True, each tweet will include a node called "entities". This node offers a variety of metadata about the tweet in a discreet structure, including: user_mentions, urls, and hashtags. While entities are opt-in on timelines at present, they will be made a default component of output in the future. See Tweet Entities for more details.
- **skip_status** -- (bool - optional) When set to Truestatuses will not be included in the returned user objects.

https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/api-reference/get-lists-subscribers

get_list_subscriptions(*user_id=None*, *screen_name=None*, *count=None*, *cursor=None*)

Obtain a collection of the lists the specified user is subscribed to.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results. Helpful for disambiguating when a valid user ID is also a valid screen name.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results. Helpful for disambiguating when a valid screen name is also a user ID.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **cursor** -- (cursor - optional) Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned in the response body's next_cursor and previous_cursor attributes to page back and forth in the list. It is recommended to always use cursors when the method supports them. See Cursoring for more information.

https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/api-reference/get-lists-subscriptions

get_mentions_timeline(*count=None*, *since_id=None*, *max_id=None*, *trim_user=None*, *include_entities=None*, *tweet_mode=None*)

Returns the 20 most recent mentions (tweets containing a users's @screen_name) for the authenticating user.

Parameters

- **count** -- (int - optional) Specifies the number of results to retrieve.
- **since_id** -- (int - optional) Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.

- **max_id** -- (int - optional) Returns results with an ID less than (that is, older than) or equal to the specified ID.
- **trim_user** -- (bool - optional) When set to True, each tweet returned in a timeline will include a user object including only the status authors numerical ID. Omit this parameter to receive the complete user object.
- **include_entities** -- (bool - optional) The entities node will not be included when set to False.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-mentions_timeline

get_place_trends(*ids*, *exclude=None*)

Returns the top 10 trending topics for a specific WOEID, if trending information is available for it.

Parameters

- **id** -- (int or list of ints - required) run `get_available_trends()` for the full listing. The Yahoo! Where On Earth ID of the location to return trending information for. Global information is available by using 1 as the WOEID .
- **exclude** -- (str - optional) Setting this equal to hashtags will remove all hashtags from the trends list.

https://developer.twitter.com/en/docs/trends/trends-for-location/api-reference/get-trends-place

get_retweeters_ids(*id*, *count=None*, *cursor=None*, *stringify_ids=None*)

Returns a collection of up to 100 user IDs belonging to users who have retweeted the tweet specified by the `id` parameter. It's better to use get_retweets because passing a count > 100 will only get you duplicated data. 100 is the maximum even if there were more retweeters.

Parameters

- **id** -- (int - required) The numerical ID of the desired status.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **cursor** -- (cursor - semi-optional) Causes the list of IDs to be broken into pages of no more than 100 IDs at a time. The number of IDs returned is not guaranteed to be 100 as suspended users are filtered out after connections are queried. If no cursor is provided, a value of -1 will be assumed, which is the first "page." The response from the API will include a previous_cursor and next_cursor to allow paging back and forth. See our cursor docs for more information. While this method supports the cursor parameter, the entire result set can be returned in a single cursored collection. Using the count parameter with this method will not provide segmented cursors for use with this parameter.
- **stringify_ids** -- (bool - optional) Many programming environments will not consume Tweet ids due to their size. Provide this option to have ids returned as strings instead.

https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-statuses-retweeters-ids

get_retweets(*id*, *trim_user=None*, *tweet_mode=None*)

Returns up to 100 of the first retweets of a given tweet.

Parameters

- **id** -- (int - required) The numerical ID of the desired status.
- **trim_user** -- (bool - optional) When set to True, each tweet returned in a timeline will include a user object including only the status authors numerical ID. Omit this parameter to receive the complete user object.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/post-statuses-retweet-id

`get_supported_languages()`

Returns the list of languages supported by Twitter along with their ISO 639-1 code.

https://developer.twitter.com/en/docs/developer-utilities/supported-languages/api-reference/get-help-languages

`get_user_timeline(`*user_id=None*, *screen_name=None*, *since_id=None*, *count=None*, *max_id=None*, *trim_user=None*, *exclude_replies=None*, *include_rts=None*, *tweet_mode=None*`)`

Returns a collection of the most recent Tweets posted by the user indicated by the `screen_name` or `user_id` parameters.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results.
- **since_id** -- (int - optional) Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets that can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **max_id** -- (int - optional) Returns results with an ID less than (that is, older than) or equal to the specified ID.
- **trim_user** -- (bool - optional) When set to True, each Tweet returned in a timeline will include a user object including only the status authors numerical ID. Omit this parameter to receive the complete user object.
- **exclude_replies** -- (bool - optional) This parameter will prevent replies from appearing in the returned timeline. Using exclude_replies with the count parameter will mean you will receive up-to count tweets — this is because the count parameter retrieves that many Tweets before filtering out retweets and replies.
- **include_rts** -- (bool - optional) When set to False , the timeline will strip any native retweets (though they will still count toward both the maximal length of the timeline and the slice selected by the count parameter). Note: If you're using the trim_user parameter in conjunction with include_rts, the retweets will still contain a full user object.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline

lookup_status(*id*, *include_entities=None*, *trim_user=None*, *map=None*, *include_ext_alt_text=None*, *include_card_uri=None*, *tweet_mode=None*)

Returns fully-hydrated tweet objects for up to 100 tweets per request, as specified by comma-separated values passed to the `id` parameter.

Parameters

- **id** -- (int - required) A comma separated list of Tweet IDs, up to 100 are allowed in a single request.
- **include_entities** -- (bool - optional) The entities node that may appear within embedded statuses will not be included when set to False.
- **trim_user** -- (bool - optional) When set to True, each Tweet returned in a timeline will include a user object including only the status authors numerical ID. Omit this parameter to receive the complete user object.
- **map** -- (bool - optional) When using the map parameter, Tweets that do not exist or cannot be viewed by the current user will still have their key represented but with an explicitly null value paired with it
- **include_ext_alt_text** -- (bool - optional) If alt text has been added to any attached media entities, this parameter will return an ext_alt_text value in the top-level key for the media entity. If no value has been set, this will be returned as null
- **include_card_uri** -- (bool - optional) When set to True, each Tweet returned will include a card_uri attribute when there is an ads card attached to the Tweet and when that card was attached using the card_uri value.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-statuses-lookup

lookup_user(*screen_name=None*, *user_id=None*, *include_entities=None*, *tweet_mode=None*)

Returns fully-hydrated user objects for up to 100 users per request, as specified by comma-separated values passed to the `user_id` and/or `screen_name` parameters.

Parameters

- **screen_name** -- (str - optional) A comma separated list of screen names, up to 100 are allowed in a single request. You are strongly encouraged to use a POST for larger (up to 100 screen names) requests.
- **user_id** -- (int - optional) A comma separated list of user IDs, up to 100 are allowed in a single request. You are strongly encouraged to use a POST for larger requests.
- **include_entities** -- (bool - optional) The entities node that may appear within embedded statuses will not be included when set to False.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-users-lookup

make_dataframe(*func*)

retweeted_of_me(*count=None*, *since_id=None*, *max_id=None*, *trim_user=None*, *include_entities=None*, *include_user_entities=None*, *tweet_mode=None*)

Returns the most recent tweets authored by the authenticating user that have been retweeted by others.

Parameters

- **count** -- (int - optional) Specifies the number of results to retrieve.
- **since_id** -- (int - optional) Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
- **max_id** -- (int - optional) Returns results with an ID less than (that is, older than) or equal to the specified ID.
- **trim_user** -- (bool - optional) When set to True, each tweet returned in a timeline will include a user object including only the status authors numerical ID. Omit this parameter to receive the complete user object.
- **include_entities** -- (bool - optional) The tweet entities node will not be included when set to False .
- **include_user_entities** -- (bool - optional) The user entities node will not be included when set to False .
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-statuses-retweets_of_me

search(*q*, *geocode=None*, *lang=None*, *locale=None*, *result_type=None*, *count=None*, *until=None*, *since_id=None*, *max_id=None*, *include_entities=None*, *tweet_mode=None*)

Returns a collection of relevant Tweets matching a specified query.

Parameters

- **q** -- (str - required) A UTF-8, URL-encoded search query of 500 characters maximum, including operators. Queries may additionally be limited by complexity.
- **geocode** -- (lat lon dist - optional) Returns tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by " latitude,longitude,radius ", where radius units must be specified as either " mi " (miles) or " km " (kilometers). Note that you cannot use the near operator via the API to geocode arbitrary locations; however you can use this geocode parameter to search near geocodes directly. A maximum of 1,000 distinct "sub- regions" will be considered when using the radius modifier.
- **lang** -- (str - optional) Restricts tweets to the given language, given by an ISO 639-1 code. Language detection is best-effort.
- **locale** -- (str - optional) Specify the language of the query you are sending (only ja is currently effective). This is intended for language- specific consumers and the default should work in the majority of cases.
- **result_type** -- (str - optional) Optional. Specifies what type of search results you would prefer to receive. The current default is "mixed." Valid values include: * mixed : Include

both popular and real time results in the response. * recent : return only the most recent results in the response * popular : return only the most popular results in the response.

- **count** -- (int - optional) Specifies the number of results to retrieve.
- **until** -- (date - optional) Returns tweets created before the given date. Date should be formatted as YYYY-MM-DD. Keep in mind that the search index has a 7-day limit. In other words, no tweets will be found for a date older than one week.
- **since_id** -- (int - optional) Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
- **max_id** -- (int - optional) Returns results with an ID less than (that is, older than) or equal to the specified ID.
- **include_entities** -- (bool - optional) The entities node will not be included when set to False.
- **tweet_mode** -- (str - optional) Valid request values are compat and extended, which give compatibility mode and extended mode, respectively for Tweets that contain over 140 characters

Operator	Finds Tweets...
watching now	containing both “watching” and “now”. This is the default operator.
“happy hour”	containing the exact phrase “happy hour”.
love OR hate	containing either “love” or “hate” (or both).
beer -root	containing “beer” but not “root”.
#haiku	containing the hashtag “haiku”.
from:interior	sent from Twitter account “interior”.
list:NASA/astronauts-in-space-now	sent from a Twitter account in the NASA list astronauts-in-space-now
to:NASA	a Tweet authored in reply to Twitter account “NASA”.
@NASA	mentioning Twitter account “NASA”.
politics filter:safe	containing “politics” with Tweets marked as potentially sensitive removed.
puppy filter:media	containing “puppy” and an image or video.
puppy -filter:retweets	containing “puppy”, filtering out retweets
puppy filter:native_video	containing “puppy” and an uploaded video, Amplify video, Periscope, or Vine.
puppy filter:periscope	containing “puppy” and a Periscope video URL.
puppy filter:vine	containing “puppy” and a Vine.
puppy filter:images	containing “puppy” and links identified as photos, including third parties such as Instagram.
puppy filter:twimg	containing “puppy” and a pic.twitter.com link representing one or more photos.
hilarious filter:links	containing “hilarious” and linking to URL.
puppy url:amazon	containing “puppy” and a URL with the word “amazon” anywhere within it.
superhero since:2015-12-21	containing “superhero” and sent since date “2015-12-21” (year-month-day).
puppy until:2015-12-21	containing “puppy” and sent before the date “2015-12-21”.
movie -scary :)	containing “movie”, but not “scary”, and with a positive attitude.
flight :(	containing “flight” and with a negative attitude.
traffic ?	containing “traffic” and asking a question.

https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets

search_users(*q*, *page=None*, *count=None*, *include_entities=None*)

Provides a simple, relevance-based search interface to public user accounts on Twitter.

Parameters

- **q** -- (str - required) The search query to run against people search.
- **page** -- (int - optional) Specifies the page of results to retrieve.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **include_entities** -- (bool - optional) The entities node will not be included in embedded Tweet objects when set to False .

https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-users-search

set_auth_params(*app_key=None*, *app_secret=None*, *oauth_token=None*, *oauth_token_secret=None*, *access_token=None*, *token_type='bearer'*, *oauth_version=1*, *api_version='1.1'*, *client_args=None*, *auth_endpoint='authenticate'*)

The main function for authentication. Needs to be called once in a session.

First you need to create a developer account and app: https://developer.twitter.com/ to get your credentials.

Different ways to authenticate: https://twython.readthedocs.io/en/latest/usage/starting_out.html

show_lists(*user_id=None*, *screen_name=None*, *reverse=None*)

Returns all lists the authenticating or specified user subscribes to, including their own.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results. Helpful for disambiguating when a valid user ID is also a valid screen name. Note: : Specifies the ID of the user to get lists from. Helpful for disambiguating when a valid user ID is also a valid screen name.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results. Helpful for disambiguating when a valid screen name is also a user ID.
- **reverse** -- (bool - optional) Set this to true if you would like owned lists to be returned first. See description above for information on how this parameter works.

https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/api-reference/get-lists-list

show_owned_lists(*user_id=None*, *screen_name=None*, *count=None*, *cursor=None*)

Returns the lists owned by the specified Twitter user.

Parameters

- **user_id** -- (int - optional) The ID of the user for whom to return results. Helpful for disambiguating when a valid user ID is also a valid screen name.
- **screen_name** -- (str - optional) The screen name of the user for whom to return results. Helpful for disambiguating when a valid screen name is also a user ID.
- **count** -- (int - optional) Specifies the number of results to retrieve.
- **cursor** -- (cursor - optional) Breaks the results into pages. Provide a value of -1 to begin paging. Provide values as returned in the response body's next_cursor and previous_cursor attributes to page back and forth in the list. It is recommended to always use cursors when the method supports them. See Cursoring for more information.

https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/api-reference/get-lists-ownerships

1.23 YouTube Data API

activities_list(*key*, *part*, *channelId=None*, *home=None*, *mine=None*, *maxResults=None*, *pageToken=None*, *publishedAfter=None*, *publishedBefore=None*, *regionCode=None*)

Returns a list of channel activity events that match the request criteria. For example, you can retrieve events associated with a particular channel or with the user's own channel.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more activity resource properties that the API response will include.If the parameter identifies a property that contains child properties, the child properties will be included in the response. For example, in an activity resource, the snippet property contains other properties that identify the type of activity, a display title for the activity, and so forth. If you set part=snippet, the API response will also contain all of those nested properties.The following list contains the part names that you can include in the parameter value and the quota cost for each part: contentDetails: 2 id: 0 snippet: 2

Filters (specify exactly one of the following parameters):

Parameters

- **channelId** -- string The channelId parameter specifies a unique YouTube channel ID. The API will then return a list of that channel's activities.
- **home** -- boolean Note: This parameter has been deprecated.For requests that set this parameter, the API response contains items similar to those that a logged-out user would see on the YouTube home page. Note that this parameter can only be used in a properly authorized request.
- **mine** -- boolean This parameter can only be used in a properly authorized request. Set this parameter's value to true to retrieve a feed of the authenticated user's activities.

Optional parameters:

Parameters

- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.
- **publishedAfter** -- datetime The publishedAfter parameter specifies the earliest date and time that an activity could have occurred for that activity to be included in the API response. If the parameter value specifies a day, but not a time, then any activities that occurred that day will be included in the result set. The value is specified in ISO 8601 (YYYY-MM-DDThh:mm:ss.sZ) format.

- **publishedBefore** -- datetime The publishedBefore parameter specifies the date and time before which an activity must have occurred for that activity to be included in the API response. If the parameter value specifies a day, but not a time, then any activities that occurred that day will be excluded from the result set. The value is specified in ISO 8601 (YYYY-MM-DDThh:mm:ss.sZ) format.
- **regionCode** -- string The regionCode parameter instructs the API to return results for the specified country. The parameter value is an ISO 3166-1 alpha-2 country code. YouTube uses this value when the authorized user's previous activity on YouTube does not provide enough information to generate the activity feed.

captions_list(*key*, *part*, *videoId*, *id=None*, *onBehalfOfContentOwner=None*)

Returns a list of caption tracks that are associated with a specified video. Note that the API response does not contain the actual captions and that the captions.download method provides the ability to retrieve a caption track.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies the caption resource parts that the API response will include.The list below contains the part names that you can include in the parameter value and the quota cost for each part: id: 0 snippet: 1
- **videoId** -- string The videoId parameter specifies the YouTube video ID of the video for which the API should return caption tracks.

Optional parameters:

Parameters

- **id** -- string The id parameter specifies a comma-separated list of IDs that identify the caption resources that should be retrieved. Each ID must identify a caption track associated with the specified video.
- **onBehalfOfContentOwner** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The actual CMS account that the user authenticates with must be linked to the specified YouTube content owner.

channel_sections_list(*key*, *part*, *channelId=None*, *id=None*, *mine=None*, *hl=None*, *onBehalfOfContentOwner=None*)

Returns a list of resources that match the API request criteria.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more channelSection resource properties that the API response will include.If the parameter identifies a property that contains child properties, the child properties will be included in the response. For example, in a channelSection resource, the snippet property contains other properties, such as a display title for the section. If you set part=snippet, the API response will also

contain all of those nested properties.The following list contains the part names that you can include in the parameter value and the quota cost for each part: contentDetails: 2 id: 0 localizations: 2 snippet: 2 targeting: 2

Filters (specify exactly one of the following parameters):

Parameters

- **channelId** -- string The channelId parameter specifies a YouTube channel ID. If a request specifies a value for this parameter, the API will only return the specified channel's sections.
- **id** -- string The id parameter specifies a comma-separated list of IDs that uniquely identify the channelSection resources that are being retrieved. In a channelSection resource, the id property specifies the section's ID.
- **mine** -- boolean This parameter can only be used in a properly authorized request. Set this parameter's value to true to retrieve a feed of the channel sections associated with the authenticated user's YouTube channel.

Optional parameters:

Parameters

- **hl** -- string The hl parameter instructs the API to retrieve localized resource metadata for a specific application language that the YouTube website supports. The parameter value must be a language code included in the list returned by the i18nLanguages.list method.If localized resource details are available in that language, the resource's snippet.localized object will contain the localized values. However, if localized details are not available, the snippet.localized object will contain resource details in the resource's default language.
- **onBehalfOfContentOwner** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.

channels_list(*key*, *part*, *categoryId=None*, *forUsername=None*, *id=None*, *managedByMe=None*, *mine=None*, *mySubscribers=None*, *hl=None*, *maxResults=None*, *onBehalfOfContentOwner=None*, *pageToken=None*)

Returns a collection of zero or more resources that match the request criteria.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more channel resource properties that the API response will include.If the parameter identifies a property that contains child properties, the child properties will be included in the response. For example, in a channel resource, the contentDetails property contains other properties, such as the uploads properties. As such, if you set part=contentDetails, the API response will also contain all of those nested properties.The following list contains the part names that you can include in the parameter value and the quota cost for each part: auditDetails: 4 brandingSettings: 2 contentDetails: 2 contentOwnerDetails: 2 id: 0 invideoPromotion: 2 (deprecated) localizations: 2 snippet: 2 statistics: 2 status: 2 topicDetails: 2

Filters (specify exactly one of the following parameters):

Parameters

- **categoryId** -- string The categoryId parameter specifies a YouTube guide category, thereby requesting YouTube channels associated with that category.
- **forUsername** -- string The forUsername parameter specifies a YouTube username, thereby requesting the channel associated with that username.
- **id** -- string The id parameter specifies a comma-separated list of the YouTube channel ID(s) for the resource(s) that are being retrieved. In a channel resource, the id property specifies the channel's YouTube channel ID.
- **managedByMe** -- boolean This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.Set this parameter's value to true to instruct the API to only return channels managed by the content owner that the onBehalfOfContentOwner parameter specifies. The user must be authenticated as a CMS account linked to the specified content owner and onBehalfOfContentOwner must be provided.
- **mine** -- boolean This parameter can only be used in a properly authorized request. Set this parameter's value to true to instruct the API to only return channels owned by the authenticated user.
- **mySubscribers** -- boolean This parameter has been deprecated. This parameter can only be used in a properly authorized request. Use the subscriptions.list method and its mySubscribers parameter to retrieve a list of subscribers to the authenticated user's channel.

Optional parameters:

Parameters

- **hl** -- string The hl parameter instructs the API to retrieve localized resource metadata for a specific application language that the YouTube website supports. The parameter value must be a language code included in the list returned by the i18nLanguages.list method.If localized resource details are available in that language, the resource's snippet.localized object will contain the localized values. However, if localized details are not available, the snippet.localized object will contain resource details in the resource's default language.
- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **onBehalfOfContentOwner** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.

comment_threads_list(*key*, *part*, *allThreadsRelatedToChannelId=None*, *channelId=None*, *id=None*, *videoId=None*, *maxResults=None*, *moderationStatus=None*, *order=None*, *pageToken=None*, *searchTerms=None*, *textFormat=None*)

Returns a list of comment threads that match the API request parameters.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more commentThread resource properties that the API response will include.The following list contains the part names that you can include in the parameter value and the quota cost for each part: id: 0 replies: 2 snippet: 2

Filters (specify exactly one of the following parameters):

Parameters

- **allThreadsRelatedToChannelId** -- string The allThreadsRelatedToChannelId parameter instructs the API to return all comment threads associated with the specified channel. The response can include comments about the channel or about the channel's videos.
- **channelId** -- string The channelId parameter instructs the API to return comment threads containing comments about the specified channel. (The response will not include comments left on videos that the channel uploaded.)
- **id** -- string The id parameter specifies a comma-separated list of comment thread IDs for the resources that should be retrieved.
- **videoId** -- string The videoId parameter instructs the API to return comment threads associated with the specified video ID.

Optional parameters:

Parameters

- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **moderationStatus** -- string This parameter can only be used in a properly authorized request. Set this parameter to limit the returned comment threads to a particular moderation state.Note: This parameter is not supported for use in conjunction with the id parameter. The default value is published.Acceptable values are: heldForReview – Retrieve comment threads that are awaiting review by a moderator. A comment thread can be included in the response if the top-level comment or at least one of the replies to that comment are awaiting review. likelySpam – Retrieve comment threads classified as likely to be spam. A comment thread can be included in the response if the top-level comment or at least one of the replies to that comment is considered likely to be spam. published – Retrieve threads of published comments. This is the default value. A comment thread can be included in the response if its top-level comment has been published.
- **order** -- string The order parameter specifies the order in which the API response should list comment threads. Valid values are: time - Comment threads are ordered by time. This is the default behavior.relevance - Comment threads are ordered by relevance.Note: This parameter is not supported for use in conjunction with the id parameter.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken property identifies the next page of the result that can be retrieved.Note: This parameter is not supported for use in conjunction with the id parameter.
- **searchTerms** -- string The searchTerms parameter instructs the API to limit the API response to only contain comments that contain the specified search terms.Note: This parameter is not supported for use in conjunction with the id parameter.

- **textFormat** -- string Set this parameter's value to html or plainText to instruct the API to return the comments left by users in html formatted or in plain text. The default value is html.Acceptable values are: html – Returns the comments in HTML format. This is the default value. plainText – Returns the comments in plain text format.

comments_list(*key*, *part*, *id=None*, *parentId=None*, *maxResults=None*, *pageToken=None*, *textFormat=None*)

Returns a list of comments that match the API request parameters.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more comment resource properties that the API response will include.The following list contains the part names that you can include in the parameter value and the quota cost for each part: id: 0 snippet: 1

Filters (specify exactly one of the following parameters):

Parameters

- **id** -- string The id parameter specifies a comma-separated list of comment IDs for the resources that are being retrieved. In a comment resource, the id property specifies the comment's ID.
- **parentId** -- string The parentId parameter specifies the ID of the comment for which replies should be retrieved. Note: YouTube currently supports replies only for top-level comments. However, replies to replies may be supported in the future.

Optional parameters:

Parameters

- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken property identifies the next page of the result that can be retrieved.Note: This parameter is not supported for use in conjunction with the id parameter.
- **textFormat** -- string This parameter indicates whether the API should return comments formatted as HTML or as plain text. The default value is html.Acceptable values are: html – Returns the comments in HTML format. This is the default value. plainText – Returns the comments in plain text format.

guide_categories_list(*key*, *part*, *id=None*, *regionCode=None*, *hl=None*)

Returns a list of categories that can be associated with YouTube channels.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies the guideCategory resource properties that the API response will include. Set the parameter value to snippet. The snippet part has a quota cost of 2 units.

Filters (specify exactly one of the following parameters):

Parameters

- **id** -- string The id parameter specifies a comma-separated list of the YouTube channel category ID(s) for the resource(s) that are being retrieved. In a guideCategory resource, the id property specifies the YouTube channel category ID.
- **regionCode** -- string The regionCode parameter instructs the API to return the list of guide categories available in the specified country. The parameter value is an ISO 3166-1 alpha-2 country code.

Optional parameters:

Parameters **hl** -- string The hl parameter specifies the language that will be used for text values in the API response. The default value is en-US.

i18n_languages_list(*key*, *part*, *hl=None*)

Returns a list of application languages that the YouTube website supports.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies the i18nLanguage resource properties that the API response will include. Set the parameter value to snippet. The snippet part has a quota cost of 1 unit.

Optional parameters:

Parameters **hl** -- string The hl parameter specifies the language that should be used for text values in the API response. The default value is en_US.

i18n_regions_list(*key*, *part*, *hl=None*)

Returns a list of content regions that the YouTube website supports.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies the i18nRegion resource properties that the API response will include. Set the parameter value to snippet. The snippet part has a quota cost of 1 unit.

Optional parameters:

Parameters **hl** -- string The hl parameter specifies the language that should be used for text values in the API response. The default value is en_US.

playlist_items_list(*key*, *part*, *id=None*, *playlistId=None*, *maxResults=None*, *onBehalfOfContentOwner=None*, *pageToken=None*, *videoId=None*)

Returns a collection of playlist items that match the API request parameters. You can retrieve all of the playlist items in a specified playlist or retrieve one or more playlist items by their unique IDs.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more playlistItem resource properties that the API response will include.If the parameter identifies a property that contains child properties, the child properties will be included in the response. For example, in a playlistItem resource, the snippet property contains numerous fields, including

the title, description, position, and resourceId properties. As such, if you set part=snippet, the API response will contain all of those properties.The following list contains the part names that you can include in the parameter value and the quota cost for each part: contentDetails: 2 id: 0 snippet: 2 status: 2

Filters (specify exactly one of the following parameters):

Parameters

- **id** -- string The id parameter specifies a comma-separated list of one or more unique playlist item IDs.
- **playlistId** -- string The playlistId parameter specifies the unique ID of the playlist for which you want to retrieve playlist items. Note that even though this is an optional parameter, every request to retrieve playlist items must specify a value for either the id parameter or the playlistId parameter.

Optional parameters:

Parameters

- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **onBehalfOfContentOwner** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.
- **videoId** -- string The videoId parameter specifies that the request should return only the playlist items that contain the specified video.

playlists_list(*key*, *part*, *channelId=None*, *id=None*, *mine=None*, *hl=None*, *maxResults=None*, *onBehalfOfContentOwner=None*, *onBehalfOfContentOwnerChannel=None*, *pageToken=None*)

Returns a collection of playlists that match the API request parameters. For example, you can retrieve all playlists that the authenticated user owns, or you can retrieve one or more playlists by their unique IDs.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more playlist resource properties that the API response will include.If the parameter identifies a property that contains child properties, the child properties will be included in the response. For example, in a playlist resource, the snippet property contains properties like author, title, description, tags, and timeCreated. As such, if you set part=snippet, the API response will contain all of those properties.The following list contains the part names that you can include in the parameter value and the quota cost for each part: contentDetails: 2 id: 0 localizations: 2 player: 0 snippet: 2 status: 2

Filters (specify exactly one of the following parameters):

Parameters

- **channelId** -- string This value indicates that the API should only return the specified channel's playlists.
- **id** -- string The id parameter specifies a comma-separated list of the YouTube playlist ID(s) for the resource(s) that are being retrieved. In a playlist resource, the id property specifies the playlist's YouTube playlist ID.
- **mine** -- boolean This parameter can only be used in a properly authorized request. Set this parameter's value to true to instruct the API to only return playlists owned by the authenticated user.

Optional parameters:

Parameters

- **hl** -- string The hl parameter instructs the API to retrieve localized resource metadata for a specific application language that the YouTube website supports. The parameter value must be a language code included in the list returned by the i18nLanguages.list method.If localized resource details are available in that language, the resource's snippet.localized object will contain the localized values. However, if localized details are not available, the snippet.localized object will contain resource details in the resource's default language.
- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **onBehalfOfContentOwner** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.
- **onBehalfOfContentOwnerChannel** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwnerChannel parameter specifies the YouTube channel ID of the channel to which a video is being added. This parameter is required when a request specifies a value for the onBehalfOfContentOwner parameter, and it can only be used in conjunction with that parameter. In addition, the request must be authorized using a CMS account that is linked to the content owner that the onBehalfOfContentOwner parameter specifies. Finally, the channel that the onBehalfOfContentOwnerChannel parameter value specifies must be linked to the content owner that the onBehalfOfContentOwner parameter specifies.This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and perform actions on behalf of the channel specified in the parameter value, without having to provide authentication credentials for each separate channel.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.

search(*key*, *part*, *forContentOwner=None*, *forDeveloper=None*, *forMine=None*, *relatedToVideoId=None*, *channelId=None*, *channelType=None*, *eventType=None*, *location=None*, *locationRadius=None*, *maxResults=None*, *onBehalfOfContentOwner=None*, *order=None*, *pageToken=None*, *publishedAfter=None*, *publishedBefore=None*, *q=None*, *regionCode=None*, *relevanceLanguage=None*, *safeSearch=None*, *topicId=None*, *type=None*, *videoCaption=None*, *videoCategoryId=None*, *videoDefinition=None*, *videoDimension=None*, *videoDuration=None*, *videoEmbeddable=None*, *videoLicense=None*, *videoSyndicated=None*, *videoType=None*)

Returns a collection of search results that match the query parameters specified in the API request. By default, a search result set identifies matching video , channel , and playlist resources, but you can also configure queries to only retrieve a specific type of resource.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more search resource properties that the API response will include. Set the parameter value to snippet.

Filters (specify 0 or 1 of the following parameters):

Parameters

- **forContentOwner** -- boolean This parameter can only be used in a properly authorized request, and it is intended exclusively for YouTube content partners.The forContentOwner parameter restricts the search to only retrieve videos owned by the content owner identified by the onBehalfOfContentOwner parameter. If forContentOwner is set to true, the request must also meet these requirements:The onBehalfOfContentOwner parameter is required.The user authorizing the request must be using an account linked to the specified content owner.The type parameter value must be set to video.None of the following other parameters can be set: videoDefinition, videoDimension, videoDuration, videoLicense, videoEmbeddable, videoSyndicated, videoType.
- **forDeveloper** -- boolean This parameter can only be used in a properly authorized request. The forDeveloper parameter restricts the search to only retrieve videos uploaded via the developer's application or website. The API server uses the request's authorization credentials to identify the developer. The forDeveloper parameter can be used in conjunction with optional search parameters like the q parameter.For this feature, each uploaded video is automatically tagged with the project number that is associated with the developer's application in the Google Developers Console.When a search request subsequently sets the forDeveloper parameter to true, the API server uses the request's authorization credentials to identify the developer. Therefore, a developer can restrict results to videos uploaded through the developer's own app or website but not to videos uploaded through other apps or sites.
- **forMine** -- boolean This parameter can only be used in a properly authorized request. The forMine parameter restricts the search to only retrieve videos owned by the authenticated user. If you set this parameter to true, then the type parameter's value must also be set to video. In addition, none of the following other parameters can be set in the same request: videoDefinition, videoDimension, videoDuration, videoLicense, videoEmbeddable, videoSyndicated, videoType.
- **relatedToVideoId** -- string The relatedToVideoId parameter retrieves a list of videos that are related to the video that the parameter value identifies. The parameter value must be set to a YouTube video ID and, if you are using this parameter, the type parameter must be set to video.Note that if the relatedToVideoId parameter is set, the only other supported parameters are part, maxResults, pageToken, regionCode, relevanceLanguage, safeSearch, type (which must be set to video), and fields.

Optional parameters:

Parameters

- **channelId** -- string The channelId parameter indicates that the API response should only contain resources created by the channel. Note: Search results are constrained to a maximum of 500 videos if your request specifies a value for the channelId parameter and sets the type parameter value to video, but it does not also set one of the forContentOwner, forDeveloper, or forMine filters.
- **channelType** -- string The channelType parameter lets you restrict a search to a particular type of channel.Acceptable values are: any – Return all channels. show – Only retrieve shows.
- **eventType** -- string The eventType parameter restricts a search to broadcast events. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: completed – Only include completed broadcasts. live – Only include active broadcasts. upcoming – Only include upcoming broadcasts.
- **location** -- string The location parameter, in conjunction with the locationRadius parameter, defines a circular geographic area and also restricts a search to videos that specify, in their metadata, a geographic location that falls within that area. The parameter value is a string that specifies latitude/longitude coordinates e.g. (37.42307,-122.08427).The location parameter value identifies the point at the center of the area.The locationRadius parameter specifies the maximum distance that the location associated with a video can be from that point for the video to still be included in the search results.The API returns an error if your request specifies a value for the location parameter but does not also specify a value for the locationRadius parameter.
- **locationRadius** -- string The locationRadius parameter, in conjunction with the location parameter, defines a circular geographic area.The parameter value must be a floating point number followed by a measurement unit. Valid measurement units are m, km, ft, and mi. For example, valid parameter values include 1500m, 5km, 10000ft, and 0.75mi. The API does not support locationRadius parameter values larger than 1000 kilometers.Note: See the definition of the location parameter for more information.
- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **onBehalfOfContentOwner** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.
- **order** -- string The order parameter specifies the method that will be used to order resources in the API response. The default value is relevance.Acceptable values are: date – Resources are sorted in reverse chronological order based on the date they were created. rating – Resources are sorted from highest to lowest rating. relevance – Resources are sorted based on their relevance to the search query. This is the default value for this parameter. title – Resources are sorted alphabetically by title. videoCount – Channels are sorted in descending order of their number of uploaded videos. viewCount – Resources are sorted from highest to lowest number of views. For live broadcasts, videos are sorted by number of concurrent viewers while the broadcasts are ongoing.

- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.
- **publishedAfter** -- datetime The publishedAfter parameter indicates that the API response should only contain resources created at or after the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).
- **publishedBefore** -- datetime The publishedBefore parameter indicates that the API response should only contain resources created before or at the specified time. The value is an RFC 3339 formatted date-time value (1970-01-01T00:00:00Z).
- **q** -- string The q parameter specifies the query term to search for.Your request can also use the Boolean NOT (-) and OR (|) operators to exclude videos or to find videos that are associated with one of several search terms. For example, to search for videos matching either "boating" or "sailing", set the q parameter value to boating|sailing. Similarly, to search for videos matching either "boating" or "sailing" but not "fishing", set the q parameter value to boating|sailing -fishing. Note that the pipe character must be URL-escaped when it is sent in your API request. The URL-escaped value for the pipe character is %7C.
- **regionCode** -- string The regionCode parameter instructs the API to return search results for videos that can be viewed in the specified country. The parameter value is an ISO 3166-1 alpha-2 country code.
- **relevanceLanguage** -- string The relevanceLanguage parameter instructs the API to return search results that are most relevant to the specified language. The parameter value is typically an ISO 639-1 two- letter language code. However, you should use the values zh-Hans for simplified Chinese and zh-Hant for traditional Chinese. Please note that results in other languages will still be returned if they are highly relevant to the search query term.
- **safeSearch** -- string The safeSearch parameter indicates whether the search results should include restricted content as well as standard content.Acceptable values are: moderate – YouTube will filter some content from search results and, at the least, will filter content that is restricted in your locale. Based on their content, search results could be removed from search results or demoted in search results. This is the default parameter value. none – YouTube will not filter the search result set. strict – YouTube will try to exclude all restricted content from the search result set. Based on their content, search results could be removed from search results or demoted in search results.
- **topicId** -- string The topicId parameter indicates that the API response should only contain resources associated with the specified topic. The value identifies a Freebase topic ID.Important: Due to the deprecation of Freebase and the Freebase API, the topicId parameter started working differently as of February 27, 2017. At that time, YouTube started supporting a small set of curated topic IDs, and you can only use that smaller set of IDs as values for this parameter. See topic IDs supported as of February 15, 2017 Topics Music topics /m/04rlf Music (parent topic) /m/02mscn Christian music /m/0ggq0m Classical music /m/01lyv Country /m/02lkt Electronic music /m/0glt670 Hip hop music /m/05rwpb Independent music /m/03_d0 Jazz /m/028sqc Music of Asia /m/0g293 Music of Latin America /m/064t9 Pop music /m/06cqb Reggae /m/06j6l Rhythm and blues /m/06by7 Rock music /m/0gywn Soul music Gaming topics /m/0bzvm2 Gaming (parent topic) /m/025zzc Action game /m/02ntfj Action-adventure game /m/0b1vjn Casual game /m/02hygl Music video game /m/04q1x3q Puzzle video game /m/01sjng Racing video game /m/0403l3g Role-playing video game /m/021bp2 Simulation video game /m/022dc6 Sports game /m/03hf_rm Strategy video game Sports topics /m/06ntj Sports (parent topic) /m/0jm_ American football /m/018jz Baseball /m/018w8 Basketball /m/01cgz Boxing /m/09xp_ Cricket /m/02vx4 Football /m/037hz Golf /m/03tmr Ice hockey /m/01h7lh Mixed martial arts /m/0410tth Motorsport /m/07bs0 Tennis /m/07_53 Volleyball Entertainment topics /m/02jjt Entertainment (parent

topic) /m/09kqc Humor /m/02vxn Movies /m/05qjc Performing arts /m/066wd Professional wrestling /m/0f2f9 TV shows Lifestyle topics /m/019_rr Lifestyle (parent topic) /m/032tl Fashion /m/027x7n Fitness /m/02wbm Food /m/03glg Hobby /m/068hy Pets /m/041xxh Physical attractiveness [Beauty] /m/07c1v Technology /m/07bxq Tourism /m/07yv9 Vehicles Society topics /m/098wr Society (parent topic) /m/09s1f Business /m/0kt51 Health /m/01h6rj Military /m/05qt0 Politics /m/06bvp Religion Other topics /m/01k8wb Knowledge

- **type** -- string The type parameter restricts a search query to only retrieve a particular type of resource. The value is a comma-separated list of resource types. The default value is video,channel,playlist.Acceptable values are: channelplaylistvideo
- **videoCaption** -- string The videoCaption parameter indicates whether the API should filter video search results based on whether they have captions. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: any – Do not filter results based on caption availability. closedCaption – Only include videos that have captions. none – Only include videos that do not have captions.
- **videoCategoryId** -- string The videoCategoryId parameter filters video search results based on their category. If you specify a value for this parameter, you must also set the type parameter's value to video.
- **videoDefinition** -- string The videoDefinition parameter lets you restrict a search to only include either high definition (HD) or standard definition (SD) videos. HD videos are available for playback in at least 720p, though higher resolutions, like 1080p, might also be available. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: any – Return all videos, regardless of their resolution. high – Only retrieve HD videos. standard – Only retrieve videos in standard definition.
- **videoDimension** -- string The videoDimension parameter lets you restrict a search to only retrieve 2D or 3D videos. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: 2d – Restrict search results to exclude 3D videos. 3d – Restrict search results to only include 3D videos. any – Include both 3D and non-3D videos in returned results. This is the default value.
- **videoDuration** -- string The videoDuration parameter filters video search results based on their duration. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: any – Do not filter video search results based on their duration. This is the default value. long – Only include videos longer than 20 minutes. medium – Only include videos that are between four and 20 minutes long (inclusive). short – Only include videos that are less than four minutes long.
- **videoEmbeddable** -- string The videoEmbeddable parameter lets you to restrict a search to only videos that can be embedded into a webpage. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: any – Return all videos, embeddable or not. true – Only retrieve embeddable videos.
- **videoLicense** -- string The videoLicense parameter filters search results to only include videos with a particular license. YouTube lets video uploaders choose to attach either the Creative Commons license or the standard YouTube license to each of their videos. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: any – Return all videos, regardless of which license they have, that match the query parameters. creativeCommon – Only return videos that have a Creative Commons license. Users can reuse videos with this license in other videos that they create. Learn more. youtube – Only return videos that have the standard YouTube license.
- **videoSyndicated** -- string The videoSyndicated parameter lets you to restrict a search to only videos that can be played outside youtube.com. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: any – Return all

videos, syndicated or not. true – Only retrieve syndicated videos.

- **videoType** -- string The videoType parameter lets you restrict a search to a particular type of videos. If you specify a value for this parameter, you must also set the type parameter's value to video.Acceptable values are: any – Return all videos. episode – Only retrieve episodes of shows. movie – Only retrieve movies.

subscriptions_list(*key*, *part*, *channelId=None*, *id=None*, *mine=None*, *myRecentSubscribers=None*, *mySubscribers=None*, *forChannelId=None*, *maxResults=None*, *onBehalfOfContentOwner=None*, *onBehalfOfContentOwnerChannel=None*, *order=None*, *pageToken=None*)

Returns subscription resources that match the API request criteria.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more subscription resource properties that the API response will include.If the parameter identifies a property that contains child properties, the child properties will be included in the response. For example, in a subscription resource, the snippet property contains other properties, such as a display title for the subscription. If you set part=snippet, the API response will also contain all of those nested properties.The following list contains the part names that you can include in the parameter value and the quota cost for each part: contentDetails: 2 id: 0 snippet: 2 subscriberSnippet: 2

Filters (specify exactly one of the following parameters):

Parameters

- **channelId** -- string The channelId parameter specifies a YouTube channel ID. The API will only return that channel's subscriptions.
- **id** -- string The id parameter specifies a comma-separated list of the YouTube subscription ID(s) for the resource(s) that are being retrieved. In a subscription resource, the id property specifies the YouTube subscription ID.
- **mine** -- boolean This parameter can only be used in a properly authorized request. Set this parameter's value to true to retrieve a feed of the authenticated user's subscriptions.
- **myRecentSubscribers** -- boolean This parameter can only be used in a properly authorized request. Set this parameter's value to true to retrieve a feed of the subscribers of the authenticated user in reverse chronological order (newest first).Note that this parameter only supports retrieval of the most recent 1000 subscribers to the authenticated user's channel. To retrieve a complete list of subscribers, use the mySubscribers parameter. That parameter, which does not return subscribers in a particular order, does not limit the number of subscribers that can be retrieved.
- **mySubscribers** -- boolean This parameter can only be used in a properly authorized request. Set this parameter's value to true to retrieve a feed of the subscribers of the authenticated user in no particular order.

Optional parameters:

Parameters

- **forChannelId** -- string The forChannelId parameter specifies a comma- separated list of channel IDs. The API response will then only contain subscriptions matching those channels.

- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.
- **onBehalfOfContentOwner** -- string Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.
- **onBehalfOfContentOwnerChannel** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwnerChannel parameter specifies the YouTube channel ID of the channel to which a video is being added. This parameter is required when a request specifies a value for the onBehalfOfContentOwner parameter, and it can only be used in conjunction with that parameter. In addition, the request must be authorized using a CMS account that is linked to the content owner that the onBehalfOfContentOwner parameter specifies. Finally, the channel that the onBehalfOfContentOwnerChannel parameter value specifies must be linked to the content owner that the onBehalfOfContentOwner parameter specifies.This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and perform actions on behalf of the channel specified in the parameter value, without having to provide authentication credentials for each separate channel.
- **order** -- string The order parameter specifies the method that will be used to sort resources in the API response. The default value is SUBSCRIPTION_ORDER_RELEVANCE.Acceptable values are: alphabetical – Sort alphabetically. relevance – Sort by relevance. unread – Sort by order of activity.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.

video_categories_list(*key*, *part*, *id=None*, *regionCode=None*, *hl=None*)

Returns a list of categories that can be associated with YouTube videos.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies the videoCategory resource properties that the API response will include. Set the parameter value to snippet. The snippet part has a quota cost of 2 units.

Filters (specify exactly one of the following parameters):

Parameters

- **id** -- string The id parameter specifies a comma-separated list of video category IDs for the resources that you are retrieving.
- **regionCode** -- string The regionCode parameter instructs the API to return the list of video categories available in the specified country. The parameter value is an ISO 3166-1 alpha-2 country code.

Optional parameters:

Parameters **hl** -- string The hl parameter specifies the language that should be used for text values in the API response. The default value is en_US.

videos_list(*key*, *part*, *chart=None*, *id=None*, *myRating=None*, *hl=None*, *maxHeight=None*, *maxResults=None*, *maxWidth=None*, *onBehalfOfContentOwner=None*, *pageToken=None*, *regionCode=None*, *videoCategoryId=None*)

Returns a list of videos that match the API request parameters.

Required parameters:

Parameters

- **key** -- string Your Google API key.
- **part** -- string The part parameter specifies a comma-separated list of one or more video resource properties that the API response will include.If the parameter identifies a property that contains child properties, the child properties will be included in the response. For example, in a video resource, the snippet property contains the channelId, title, description, tags, and categoryId properties. As such, if you set part=snippet, the API response will contain all of those properties.The following list contains the part names that you can include in the parameter value and the quota cost for each part: contentDetails: 2 fileDetails: 1 id: 0 liveStreamingDetails: 2 localizations: 2 player: 0 processingDetails: 1 recordingDetails: 2 snippet: 2 statistics: 2 status: 2 suggestions: 1 topicDetails: 2

Filters (specify exactly one of the following parameters):

Parameters

- **chart** -- string The chart parameter identifies the chart that you want to retrieve.Acceptable values are: mostPopular – Return the most popular videos for the specified content region and video category.
- **id** -- string The id parameter specifies a comma-separated list of the YouTube video ID(s) for the resource(s) that are being retrieved. In a video resource, the id property specifies the video's ID.
- **myRating** -- string This parameter can only be used in a properly authorized request. Set this parameter's value to like or dislike to instruct the API to only return videos liked or disliked by the authenticated user.Acceptable values are: dislike – Returns only videos disliked by the authenticated user. like – Returns only video liked by the authenticated user.

Optional parameters:

Parameters

- **hl** -- string The hl parameter instructs the API to retrieve localized resource metadata for a specific application language that the YouTube website supports. The parameter value must be a language code included in the list returned by the i18nLanguages.list method.If localized resource details are available in that language, the resource's snippet.localized object will contain the localized values. However, if localized details are not available, the snippet.localized object will contain resource details in the resource's default language.
- **maxHeight** -- unsigned integer The maxHeight parameter specifies the maximum height of the embedded player returned in the player.embedHtml property. You can use this parameter to specify that instead of the default dimensions, the embed code should use a height appropriate for your application layout. If the maxWidth parameter is also provided, the player may be shorter than the maxHeight in order to not violate the maximum width. Acceptable values are 72 to 8192, inclusive.
- **maxResults** -- unsigned integer The maxResults parameter specifies the maximum number of items that should be returned in the result set.

- **maxWidth** -- unsigned integer The maxWidth parameter specifies the maximum width of the embedded player returned in the player.embedHtml property. You can use this parameter to specify that instead of the default dimensions, the embed code should use a width appropriate for your application layout.If the maxHeight parameter is also provided, the player may be narrower than maxWidth in order to not violate the maximum height. Acceptable values are 72 to 8192, inclusive.
- **onBehalfOfContentOwner** -- string This parameter can only be used in a properly authorized request. Note: This parameter is intended exclusively for YouTube content partners.The onBehalfOfContentOwner parameter indicates that the request's authorization credentials identify a YouTube CMS user who is acting on behalf of the content owner specified in the parameter value. This parameter is intended for YouTube content partners that own and manage many different YouTube channels. It allows content owners to authenticate once and get access to all their video and channel data, without having to provide authentication credentials for each individual channel. The CMS account that the user authenticates with must be linked to the specified YouTube content owner.
- **pageToken** -- string The pageToken parameter identifies a specific page in the result set that should be returned. In an API response, the nextPageToken and prevPageToken properties identify other pages that could be retrieved.Note: This parameter is supported for use in conjunction with the myRating parameter, but it is not supported for use in conjunction with the id parameter.
- **regionCode** -- string The regionCode parameter instructs the API to select a video chart available in the specified region. This parameter can only be used in conjunction with the chart parameter. The parameter value is an ISO 3166-1 alpha-2 country code.
- **videoCategoryId** -- string The videoCategoryId parameter identifies the video category for which the chart should be retrieved. This parameter can only be used in conjunction with the chart parameter. By default, charts are not restricted to a particular category. The default value is 0.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

2.1 advertools

2.1.1 advertools package

Subpackages

advertools.cli module

advertools.code_recipes package

Submodules

Module contents

Submodules

Regular Expressions for Extracting Structured Entities

A collection of regular expressions for use in different contexts. Each one is available in two formats:

- **REGEX_RAW: (HASHTAG_RAW, MENTION_RAW, etc.) raw string only, for sharing** and combining with other regexes
- REGEX: (HASHTAG, MENTION, etc.) compiled regex, readable, and annotated

Based on Unicode database v11.0.0

URL regex from Regular Expressions Cookbook 2nd Ed. O'Reilly

URL Builders

url_utm_ga(*url*, *utm_source*, *utm_medium=None*, *utm_campaign=None*, *utm_content=None*, *utm_term=None*)

Generate a URL with UTM codes for your campaigns.

Parameters

- **url** (*str*) -- a valid URL, required
- **utm_source** (*str*) -- the referrer of the traffic (e.g. facebook, twitter)
- **utm_medium** (*str*) -- marketing medium (e.g. banner, email)
- **utm_campaign** (*str*) -- the name of the campaign (e.g. summer_promo, 20pct_off)
- **utm_content** (*str*) -- ad name / differentiator (e.g. 728x90, mpu, square_banner)
- **utm_term** (*str*) -- search terms bid on (only relevant for search campaigns)

Returns URL-encoded string for the campaign

```
>>> url_utm_ga('mysite.com', utm_source='the source')
'mysite.com?utm_source=the+source'
```

```
>>> url_utm_ga('mysite.com', utm_source='the source',
...            utm_medium='THE MEDIUM!!',
...            utm_campaign='campaign*name&^%',utm_content='728x90')
'mysite.com?utm_content=728x90&utm_campaign=campaign%2Aname%26%5E%25&utm_
↪medium=THE+MEDIUM%21%21&utm_source=the+source'
```

Module contents

Digital Marketing productivity and analysis tools.

2.1.2 Change Log - advertools

0.13.1 (2022-05-11)

- **Added**
  - Command line interface with most functions
  - Make documentation interactive for most pages using `thebe-sphinx`
- **Changed**
  - Use *np.nan* wherever there are missing values in `url_to_df`
- **Fixed**
  - Don't remove double quotes from etags when downloading XML sitemaps
  - Replace instances of `pd.DataFrame.append` with `pd.concat`, which is depracated.
  - Replace empty values with np.nan for the size column in `logs_to_df`

0.13.0 (2022-02-10)

- **Added**
  - New function `crawl_headers`: A crawler that only makes *HEAD* requests to a known list of URLs.
  - New function `reverse_dns_lookup`: A way to get host information for a large list of IP addresses concurrently.
  - New options for crawling: *exclude_url_params*, *include_url_params*, *exclude_url_regex*, and *include_url_regex* for controlling which links to follow while crawling.
- **Fixed**
  - Any `custom_settings` options given to the `crawl` function that were defined using a dictionary can now be set without issues. There was an issue if those options were not strings.
- **Changed**
  - The *skip_url_params* option was removed and replaced with the more versatile `exclude_url_params`, which accepts either `True` or a list of URL parameters to exclude while following links.

0.12.3 (2021-11-27)

- **Fixed**
  - Crawler stops when provided with bad URLs in list mode.

0.12.0,1,2 (2021-11-27)

- **Added**
  - New function `logs_to_df`: Convert a log file of any non-JSON format into a pandas DataFrame and save it to a *parquet* file. This also compresses the file to a much smaller size.
  - Crawler extracts all available `img` attributes: 'alt', 'crossorigin', 'height', 'ismap', 'loading', 'longdesc', 'referrerpolicy', 'sizes', 'src', 'srcset', 'usemap', and 'width' (excluding global HTML attributes like `style` and `draggable`).
  - New parameter for the `crawl` function `skip_url_params`: Defaults to False, consistent with previous behavior, with the ability to not follow/crawl links containing any URL parameters.
  - New column for `url_to_df` "last_dir": Extract the value in the last directory for each of the URLs.
- **Changed**
  - Query parameter columns in `url_to_df` DataFrame are now sorted by how full the columns are (the percentage of values that are not *NA*)

0.11.1 (2021-04-09)

- **Added**
  - The *nofollow* attribute for nav, header, and footer links.
- **Fixed**
  - Timeout error while downloading robots.txt files.
  - Make extracting nav, header, and footer links consistent with all links.

0.11.0 (2021-03-31)

- **Added**
  - New parameter *recursive* for `sitemap_to_df` to control whether or not to get all sub sitemaps (default), or to only get the current (sitemapindex) one.
  - New columns for `sitemap_to_df`: `sitemap_size_mb` (1 MB = 1,024x1,024 bytes), and `sitemap_last_modified` and `etag` (if available).
  - Option to request multiple robots.txt files with `robotstxt_to_df`.
  - Option to save downloaded robots DataFrame(s) to a file with `robotstxt_to_df` using the new parameter `output_file`.
  - Two new columns for `robotstxt_to_df`: `robotstxt_last_modified` and `etag` (if available).
  - Raise *ValueError* in `crawl` if `css_selectors` or `xpath_selectors` contain any of the default crawl column headers
  - New XPath code recipes for custom extraction.
  - New function `crawllogs_to_df` which converts crawl logs to a DataFrame provided they were saved while using the `crawl` function.
  - New columns in `crawl`: *viewport*, *charset*, all *h* headings (whichever is available), nav, header and footer links and text, if available.
  - Crawl errors don't stop crawling anymore, and the error message is included in the output file under a new *errors* and/or *jsonld_errors* column(s).
  - In case of having JSON-LD errors, errors are reported in their respective column, and the remainder of the page is scraped.
- **Changed**
  - Removed column prefix *resp_meta_* from columns containing it
  - Redirect URLs and reasons are separated by '@@' for consistency with other multiple-value columns
  - Links extracted while crawling are not unique any more (all links are extracted).
  - Emoji data updated with v13.1.
  - Heading tags are scraped even if they are empty, e.g. <h2></h2>.
  - Default user agent for crawling is now advertools/VERSION.
- **Fixed**
  - Handle sitemap index files that contain links to themselves, with an error message included in the final DataFrame
  - Error in robots.txt files caused by comments preceded by whitespace

  - Zipped robots.txt files causing a parsing issue
  - Crawl issues on some Linux systems when providing a long list of URLs
- **Removed**
  - Columns from the `crawl` output: *url_redirected_to*, *links_fragment*

0.10.7 (2020-09-18)

- **Added**
  - New function `knowledge_graph` for querying Google's API
  - Faster `sitemap_to_df` with threads
  - New parameter *max_workers* for `sitemap_to_df` to determine how fast it could go
  - New parameter *capitalize_adgroups* for `kw_generate` to determine whether or not to keep ad groups as is, or set them to title case (the default)
- **Fixed**
  - Remove restrictions on the number of URLs provided to `crawl`, assuming *follow_links* is set to *False* (list mode)
  - JSON-LD issue breaking crawls when it's invalid (now skipped)
- **Removed**
  - Deprecate the `youtube.guide_categories_list` (no longer supported by the API)

0.10.6 (2020-06-30)

- **Added**
  - JSON-LD support in crawling. If available on a page, JSON-LD items will have special columns, and multiple JSON-LD snippets will be numbered for easy filtering
- **Changed**
  - Stricter parsing for rel attributes, making sure they are in link elements as well
  - Date column names for `robotstxt_to_df` and `sitemap_to_df` unified as "download_date"
  - Numbering OG, Twitter, and JSON-LD where multiple elements are present in the same page, follows a unified approach: no numbering for the first element, and numbers start with "1" from the second element on. "element", "element_1", "element_2" etc.

0.10.5 (2020-06-14)

- **Added**
  - **New features for the `crawl` function:**
    - Extract canonical tags if available
    - Extract alternate *href* and *hreflang* tags if available
    - Open Graph data "og:title", "og:type", "og:image", etc.
    - Twitter cards data "twitter:site", "twitter:title", etc.

- **Fixed**
  - **Minor fixes to `robotstxt_to_df`:**
    - Allow whitespace in fields
    - Allow case-insensitive fields
- **Changed**
  - `crawl` now only supports *output_file* with the extension ".jl"
  - `word_frequency` drops *wtd_freq* and *rel_value* columns if *num_list* is not provided

0.10.4 (2020-06-07)

- **Added**
  - New function `url_to_df`, splitting URLs into their components and to a DataFrame
  - Slight speed up for `robotstxt_test`

0.10.3 (2020-06-03)

- **Added**
  - New function `robotstxt_test`, testing URLs and whether they can be fetched by certain user-agents
- **Changed**
  - Documentation main page relayout, grouping of topics, & sidebar captions
  - Various documentation clarifications and new tests

0.10.2 (2020-05-25)

- **Added**
  - User-Agent info to requests getting sitemaps and robotstxt files
  - CSS/XPath selectors support for the crawl function
  - Support for custom spider settings with a new parameter `custom_settings`
- **Fixed**
  - Update changed supported search operators and values for CSE

0.10.1 (2020-05-23)

- **Changed**
  - Links are better handled, and new output columns are available: `links_url`, `links_text`, `links_fragment`, `links_nofollow`
  - `body_text` extraction is improved by containing <p>, <li>, and elements

0.10.0 (2020-05-21)

- **Added**
  - New function `crawl` for crawling and parsing websites
  - New function `robotstxt_to_df` downloading robots.txt files into DataFrames

0.9.1 (2020-05-19)

- **Added**
  - Ability to specify robots.txt file for `sitemap_to_df`
  - Ability to retreive any kind of sitemap (news, video, or images)
  - Errors column to the returnd DataFrame if any errors occur
  - A new `sitemap_downloaded` column showing datetime of getting the sitemap
- **Fixed**
  - Logging issue causing `sitemap_to_df` to log the same action twice
  - Issue preventing URLs not ending with xml or gz from being retreived
  - Correct sitemap URL showing in the `sitemap` column

0.9.0 (2020-04-03)

- **Added**
  - New function `sitemap_to_df` imports an XML sitemap into a `DataFrame`

0.8.1 (2020-02-08)

- **Changed**
  - Column *query_time* is now named *queryTime* in the *youtube* functions
  - Handle json_normalize import from pandas based on pandas version

0.8.0 (2020-02-02)

- **Added**
  - New module *youtube* connecting to all GET requests in API
  - *extract_numbers* new function
  - *emoji_search* new function
  - *emoji_df* new variable containing all emoji as a DataFrame
- **Changed**
  - Emoji database updated to v13.0
  - *serp_goog* with expanded *pagemap* and metadata
- **Fixed**

  - *serp_goog* errors, some parameters not appearing in result df
  - *extract_numbers* issue when providing dash as a separator in the middle

0.7.3 (2019-04-17)

- **Added**
  - New function *extract_exclamations* very similar to *extract_questions*
  - New function *extract_urls*, also counts top domains and top TLDs
  - New keys to *extract_emoji*; *top_emoji_categories* & *top_emoji_sub_categories*
  - Groups and sub-groups to *emoji db*

0.7.2 (2019-03-29)

- **Changed**
  - Emoji regex updated
  - Simpler extraction of Spanish *questions*

0.7.1 (2019-03-26)

- **Fixed**
  - Missing __init__ imports.

0.7.0 (2019-03-26)

- **Added**
  - New *extract_* functions:
    * Generic *extract* used by all others, and takes arbitrary regex to extract text.
    * *extract_questions* to get question mark statistics, as well as the text of questions asked.
    * *extract_currency* shows text that has currency symbols in it, as well as surrounding text.
    * *extract_intense_words* gets statistics about, and extract words with any character repeated three or more times, indicating an intense feeling (+ve or -ve).
  - New function *word_tokenize*:
    * Used by *word_frequency* to get tokens of 1,2,3-word phrases (or more).
    * Split a list of text into tokens of a specified number of words each.
  - New stop-words from the `spaCy` package:

    current: Arabic, Azerbaijani, Danish, Dutch, English, Finnish, French, German, Greek, Hungarian, Italian, Kazakh, Nepali, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish, Turkish.

    new: Bengali, Catalan, Chinese, Croatian, Hebrew, Hindi, Indonesian, Irish, Japanese, Persian, Polish, Sinhala, Tagalog, Tamil, Tatar, Telugu, Thai, Ukrainian, Urdu, Vietnamese
- **Changed**
  - ***word_frequency* takes new parameters:**

        * *regex* defaults to words, but can be changed to anything 'S+' to split words and keep punctuation for example.
        * *sep* not longer used as an option, the above *regex* can be used instead
        * *num_list* now optional, and defaults to counts of 1 each if not provided. Useful for counting *abs_freq* only if data not available.
        * *phrase_len* the number of words in each split token. Defaults to 1 and can be set to 2 or higher. This helps in analyzing phrases as opposed to words.
    - Parameters supplied to *serp_goog* appear at the beginning of the result df
    - *serp_youtube* now contains *nextPageToken* to make paginating requests easier

0.6.0 (2019-02-11)

- **New function**
    - *extract_words* to extract an arbitrary set of words
- **Minor updates**
    - *ad_from_string* slots argument reflects new text ad lenghts
    - *hashtag* regex improved

0.5.3 (2019-01-31)

- **Fix minor bugs**
    - Handle Twitter search queries with 0 results in final request

0.5.2 (2018-12-01)

- **Fix minor bugs**
    - Properly handle requests for >50 items (*serp_youtube*)
    - Rewrite test for _dict_product
    - Fix issue with string printing error msg

0.5.1 (2018-11-06)

- **Fix minor bugs**
    - _dict_product implemented with lists
    - Missing keys in some YouTube responses

0.5.0 (2018-11-04)

- **New function *serp_youtube***
  - Query YouTube API for videos, channels, or playlists
  - Multiple queries (product of parameters) in one function call
  - Reponse looping and merging handled, one DataFrame
- *serp_goog* return Google's original error messages
- twitter responses with entities, get the entities extracted, each in a separate column

0.4.1 (2018-10-13)

- **New function *serp_goog* (based on Google CSE)**
  - Query Google search and get the result in a DataFrame
  - Make multiple queries / requests in one function call
  - All responses merged in one DataFrame
- twitter.get_place_trends results are ranked by town and country

0.4.0 (2018-10-08)

- **New Twitter module based on twython**
  - Wraps 20+ functions for getting Twitter API data
  - Gets data in a pands DataFrame
  - Handles looping over requests higher than the defaults
- Tested on Python 3.7

0.3.0 (2018-08-14)

- Search engine marketing cheat sheet.
- **New set of extract_ functions with summary stats for each:**
  - extract_hashtags
  - extract_mentions
  - extract_emoji
- Tests and bug fixes

0.2.0 (2018-07-06)

- New set of kw_<match-type> functions.
- Full testing and coverage.

0.1.0 (2018-07-02)

- First release on PyPI.
- **Functions available:**
  - ad_create: create a text ad place words in placeholders
  - **ad_from_string: split a long string to shorter string that fit into** given slots
  - kw_generate: generate keywords from lists of products and words
  - url_utm_ga: generate a UTM-tagged URL for Google Analytics tracking
  - **word_frequency: measure the absolute and weighted frequency of words in** collection of documents

PYTHON MODULE INDEX

a

INDEX

A

C

E

G

H

I

K

L

M

N

P

R

S

T

U

V

W

Y