fetch-data

Release 0.2.5.2.dev3+gda69701.d20210709

Luke Gregor

Jul 09, 2021

USAGE EXAMPLES

Fetch data is a simple tool to quickly download data from various sources. It is a package I've developed for my own needs, so I haven't written too many tests. The `fetch-data` relies on two established packages, `pooch` and `fsspec`. I have combined these to make the downloading process quick and easy.

CHAPTER

ONE

INSTALLATION

```
pip install fetch-data
```

CHAPTER

TWO

CONTENTS

You can download using the straight forward, interactive case. This makes getting a file pretty easy. You could also check out `pooch` for this purpose (which `fetch_data` is built around).

2.1 Simple case

[2]:
```
import fetch_data as fd

url = (
    "https://www.ncei.noaa.gov/thredds/fileServer/OisstBase/NetCDF/V2.0/AVHRR/198111/
↪avhrr-only-v2.19811101.nc",
    "https://www.ncei.noaa.gov/thredds/fileServer/OisstBase/NetCDF/V2.0/AVHRR/198111/
↪avhrr-only-v2.19811102.nc"
)

flist = fd.download(url, dest='~/Downloads', n_jobs=1, verbose=False)

print('\n'.join(flist))
```

```
/Users/luke/Downloads/avhrr-only-v2.19811101.nc
/Users/luke/Downloads/avhrr-only-v2.19811102.nc
```

2.2 Using wildcards

You can also use wild card allocation if you want to download multiple files from a server. Note that the server needs to allow this (especially for HTTP). In this case, the we get three files from an FTP server. You can also use this when the file name is not consistent.

[3]:
```
url = "ftp://ftp.cdc.noaa.gov/Datasets/noaa.oisst.v2/sst.ltm.*.nc"
flist = fd.download(url, dest='~/Downloads', n_jobs=1, verbose=False)

print('\n'.join(flist))
```

```
/Users/luke/Downloads/sst.ltm.1961-1990.nc
/Users/luke/Downloads/sst.ltm.1971-2000.nc
/Users/luke/Downloads/sst.ltm.1981-2010.nc
```

2.3 Downloading compressed files

Compressed files are automatically decompressed. This is done automatically based on the file extension. The extensions currently supported are: `zip, gz, tar`.

Below you can see what the output looks like.

[5]:
```
url = "https://www.metoffice.gov.uk/hadobs/en4/data/en4-2-1/EN.4.2.1.analyses.g10.2001.
↪zip"
flist = fd.download(url, dest='~/Downloads', verbose=False)

print('\n'.join(flist))
```

```
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200104.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200110.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200111.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200101.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200105.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200108.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200109.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200112.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200102.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200106.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200107.nc
/Users/luke/Downloads/EN.4.2.1.analyses.g10.2001.zip.unzip/EN.4.2.1.f.analysis.g10.
↪200103.nc
```

2.4 YAML catalog

The advantage of using a catalog, is that all the information that you need to download the file is stored in a single file. This approach is better for a data pipeline approach.

Below is the format that your `yaml` file should take:

```yaml
# the name of the variable goes here.
oisst_ice:
  # url is compulsory. Can use formatting as shown below, but has to be given as a kwarg
  url: ftp://ftp2.psl.noaa.gov/Datasets/noaa.oisst.v2.highres/icec.day.mean.{year}.nc
  # will default to ~/Downloads if not present.
  dest: ~/Downloads/NOAA_OISST/{year}/
  # name
```

(continues on next page)

(continued from previous page)

```
  name: NOAA Optimally Interpolated Sea Surface Temperature
  meta:  # all entries in the meta will be written to README.txt file in the dest
    description: >
      Optimally interpolated sea surface temperature
    citation: >
      Reynolds, R.W., N.A. Rayner, T.M. Smith, D.C. Stokes, and W. Wang,
      2002: An improved in situ and satellite SST analysis for climate.
      J. Climate, 15, 1609-1625.
    doi: https://doi.org/10.1175/1520-0442(2002)015%3C1609:AIISAS%3E2.0.CO;2
```

[18]:
```
import fetch_data as fd

cat = fd.read_catalog('../tests/example_catalog.yml')
flist = fd.download(**cat['oisst_ice'], year=2000)

print('\n'.join(flist))
```

```
/Users/luke/Git/fetch-data/docs/tests/downloads/oisstv2/icec.day.mean.2000.nc
```

2.5 Logging

fetch_data also does logging to your session and/or to a file. This is useful if you want to track the progress of downloading many files. It may also be useful when some files fail to download (these are recorded).

[19]:
```
url = (
    "https://www.ncei.noaa.gov/thredds/fileServer/OisstBase/NetCDF/V2.0/AVHRR/198111/
↪avhrr-only-v2.19811101.nc",
    "https://www.ncei.noaa.gov/thredds/fileServer/OisstBase/NetCDF/V2.0/AVHRR/198111/
↪avhrr-only-v2.19811102.nc"
)

flist = fd.download(url, dest='~/Downloads', n_jobs=1, verbose=True)
```

```
2021-04-09 22:53:37 [DOWNLOAD] ␣
↪================================================================================


2021-04-09 22:53:37 [DOWNLOAD]  Start of logging session
2021-04-09 22:53:37 [DOWNLOAD]  ------------------------------------------------------------
↪----------------------
2021-04-09 22:53:37 [DOWNLOAD]    2 files at https://www.ncei.noaa.gov/thredds/
↪fileServer/OisstBase/NetCDF/V2.0/AVHRR/198111/avhrr-only-v2.19811101.nc
2021-04-09 22:53:37 [DOWNLOAD]  Files will be saved to /Users/luke/Downloads
2021-04-09 22:53:37 [DOWNLOAD]  retrieving https://www.ncei.noaa.gov/thredds/fileServer/
↪OisstBase/NetCDF/V2.0/AVHRR/198111/avhrr-only-v2.19811101.nc
2021-04-09 22:53:37 [DOWNLOAD]  retrieving https://www.ncei.noaa.gov/thredds/fileServer/
↪OisstBase/NetCDF/V2.0/AVHRR/198111/avhrr-only-v2.19811102.nc
2021-04-09 22:53:37 [DOWNLOAD]  SUMMARY: Retrieved=2, Failed=0 listing failed below:
```

[ ]:

2.6 Catalog

Contains functions relating to reading in catalog files (YAML) and ensuring that the entries are complete with metadata

fetch_data.catalog.**read_catalog**(*catalog_name*)

Used to read YAML files that contain download information. Placeholders for ENV names can also be used. See dotenv documentation for more info. The yaml files are structured as shown below

```
url: remote path to file/s. Can contain *
dest: path where the file/s will be stored (supports ~)
meta: # meta will be written to README.txt
    doi: url to the data source
    description: info about the data
    citation: how to cite this dataset
```

Parameters **catalog_name** (*str*) – the path to the catalog

Returns

a dictionary with catalog entries that is displayed as a YAML file. Can be viewed as a dictionary with the `dict` method.

Return type *YAMLdict*

class fetch_data.catalog.**YAMLdict**

Bases: *dict*

A class that displays a dictionary in YAML format. The object is still a dictionary, it is just the representation that is displayed as a YAML dictionary. This makes it useful to create your own catalogs. You can use the method YAMLdict.dict to view the object in dictionary representation

Attributes

dict returns a dictionary representation

Methods

clear()	
copy()	
fromkeys(iterable[, value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	

continues on next page

Table 1 – continued from previous page

pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(/)	Remove and return a (key, value) pair as a 2-tuple.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

property dict
returns a dictionary representation

2.7 Download

fetch_data.core.**download**(*url=''*, *login={}*, *dest='./'*, *n_jobs=1*, *use_cache=True*, *cache_name='_urls_{hash}.cache'*, *verbose=False*, *log_name='_downloads.log'*, *decompress=True*, *create_readme=True*, *readme_name='README.md'*, ***kwargs*)

Core function to fetch data from a url with a wildcard or as a list.

Allows for parallel download of data that can be set with a single url containing a wild card character or a list of urls. If the wild card is used, file names will be cached. A README.txt file will automatically be generated in *dest*, along with a downloading log, and a url cachce (if url is a string).

`download` is a Frankenstein mashup of `fsspec` and `pooch` to fetch files. It is tricky to download password protected files with `fsspec` and `pooch` does not allow for wildcard listed downloads. If the url input is only a list, `fsspec` will not be used and only `pooch`. But you can still download in parallel with this script.

Parameters

- **url** (*str, list*) – URL/s to be downloaded. If URL is a string and contains a wildcard (*), will try to search for files on the server. But this might not be possible with some HTTP websites. Caching will be used in this case. Will fail if no files could be fetched from the server.
- **login** (*dict*) – required if `username` and `passwords` are required for protocol
- **dest** (*str*) – where the files will be saved to. String formatting supported (as with url)
- **n_jobs** (*int*) – the number of parallel downloads. Will not show progress bar when n_jobs > 1. Not allowed to be larger than 8.
- **use_cache** (*bool*) – if set to True, will use cached url list instead of fetching a new list. This is useful for updating data
- **cache_name** (*str*) – the file name to which data will be cached. This file is stored relative to `dest`. The file is a simple text file showing a url for each line. This will not be used if a list is passed to url.
- **verbose** (*bool / int*) – if verbose is False, logging level set to ERROR (40) if verbose is True, logging level set to 15 if verbose is intiger, then sets logging level directly. See the logging module for more information.
- **log_name** (*str*) – the file name to which logging will be saved. The file is stored relative to `dest`. Logging level can be set with the `verbose` arg.

- **create_readme** (*bool*) – will create a readme in the destination folder
- **readme_name** (*str*) – default readme file name. can change the path relative to dest
- **kwargs** (*key=value*) – are keyword replacements for any values set in the url (if url is no a list) and dest strings

Returns a flattened list of file paths to where the data has been downloaded. If inputs are compressed, the names of the uncompressed files will be given.

Return type list

fetch_data.core.**get_url_list**(*url*, *username=None*, *password=None*, *use_cache=True*, *cache_path='./_urls_{hash}.cache'*, ***kwargs*)

If a url has a wildcard (*) value, remote files will be searched.

Leverages off the *fsspec* package. This doesn't work for all HTTP urls.

Parameters

- **url** (*str*) – If a url has a wildcard (*) value, remote files will be searched for
- **username** (*str*) – if required for given url and protocol (e.g. FTP)
- **password** (*str*) – if required for given url and protocol (e.g. FTP)
- **cache_path** (*str*) – the path where the cached files will be stored. Has a special case where *{hash}* will be replaced with a hash based on the URL.
- **use_cache** (*bool*) – if there is a file with cached remote urls, then those values will be returned as a list

Returns a sorted list of urls

Return type list

fetch_data.core.**download_urls**(*urls*, *downloader=None*, *n_jobs=8*, *dest_dir='.'*, *login={}*, *decompress=True*, ***kwargs*)

Downloads the given list of urls to a specified destination path using the *pooch* package in Python. NOTE: *fsspec* is not used as it fails for some FTP and SFTP protocols.

Parameters

- **urls** (*list*) – the list of URLS to download - may not contain wildcards
- **dest_dir** (*str*) – the location where the files will be downloaded to. May contain
- **formatters that are labelled with "{t** (*date*) – %fmt} to create subfolders
- **date_format** (*str*) – the format of the date in the urls that will be used to
- **in the date formatters in dest_dir kwarg. Matches limited to** (*fill*) –
- **to 2020s** (*1970s*) –
- **kwargs** (*key=value*) – will be passed to pooch.retrieve. Can be used to set
- **downloader with username and password and the processor for unzipping.** (*the*) –
- **choose_downloader for more info.** (*See*) –

Returns file names of downloaded urls

Return type list

fetch_data.core.**choose_downloader**(*url*, *login={}*, *progress=True*)
Will automatically select the correct downloader for the given url. Pass result to pooch.retrieve(downloader=downloader())

Parameters

- **url** (*str*) – the path of a url
- **login** (*dict*) – can contain either *username* and *password* OR *cookies* which are passed to the relevant downloader in pooch.
- **progress** (*bool*) – a progressbar will be shown if True - requires tqdm

Returns

with the items in login passed to the downloader as kwargs and progressbar set to True (if set)

Return type pooch.Downloader

fetch_data.core.**choose_processor**(*url*)
chooses the processor to uncompress if required

fetch_data.core.**create_download_readme**(*fname*, ***entry*)
Creates a README file based on the information in the source dictionary.

Parameters

- **name** (*str*) – name to which file will be written
- ****entry** (*kwargs*) – must contain

2.8 Utilities

Helper functions for download. Only core python packages used in utils.

fetch_data.utils.**log_to_stdout**(*level=15*)
Adds the stdout to the logging stream and sets the level to 15 by default

fetch_data.utils.**log_to_file**(*fname*)
Will append the given file path to the logger so that stdout and the file will be the output streams for the current logger

fetch_data.utils.**make_readme_file**(*dataset_name*, *url*, *meta={}*, *short_info_len_limit=150*)
Adheres to the UP group's (ETHZ) readme prerequisites.

Parameters

- **dataset_name** (*str*) – The name of the dataset that will be at the top of the file
- **url** (*str*) – The url used to download the data - may be useful for other downloaders. May contain wildcards and placeholders.
- **meta** (*dict*) – A dictionary containing several

fetch_data.utils.**make_hash_string**(*string*, *output_length=10*)
Create a hash for given string

Truncates an md5 hash to the desired length. Will always be safe for file names.

Parameters

- **string** (*str*) – input string

- **output_length** (*int*) – length for output

Returns n character string that is unique to the input string

Return type str

fetch_data.utils.**flatten_list**(*list_of_lists*)
Will recursively flatten a nested list

fetch_data.utils.**get_kwargs**()
Gets all the keyword, value pairings in the given function and returns them as a dictionary

fetch_data.utils.**abbreviate_list_as_str**(*ls*)
Abbreviates a list when it's too long to show everything

Used mostly in logging.DEBUG

fetch_data.utils.**shorten_url**(*s*, *len_limit=75*)
Make url shorter with max len set to len_limit

fetch_data.utils.**get_git_username_and_email**()
will try to get the user and email from the git config

fetch_data.utils.**commong_substring**(*input_list*)
Finds the common substring in a list of strings

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX