InstaTweet

Release 2.2.1

Adam Korn

Jun 15, 2023

INSTATWEET README

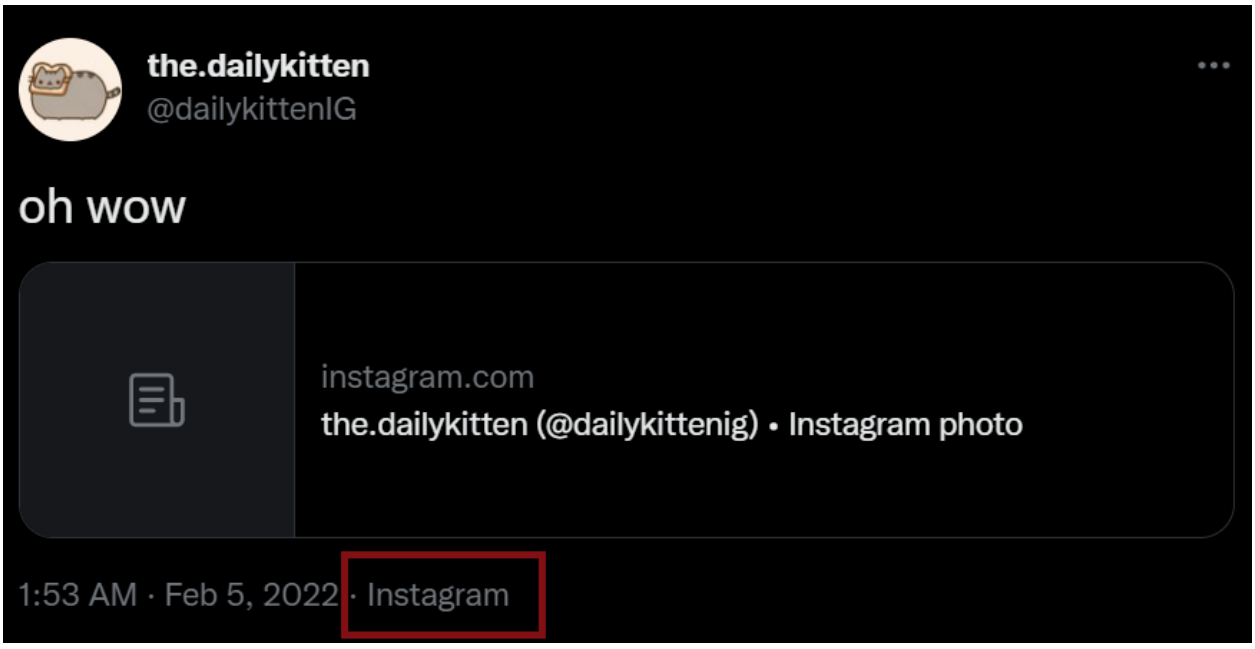
CHAPTER

ONE

INSTATWEET

1.1 Automatically Repost Content From Instagram to Twitter

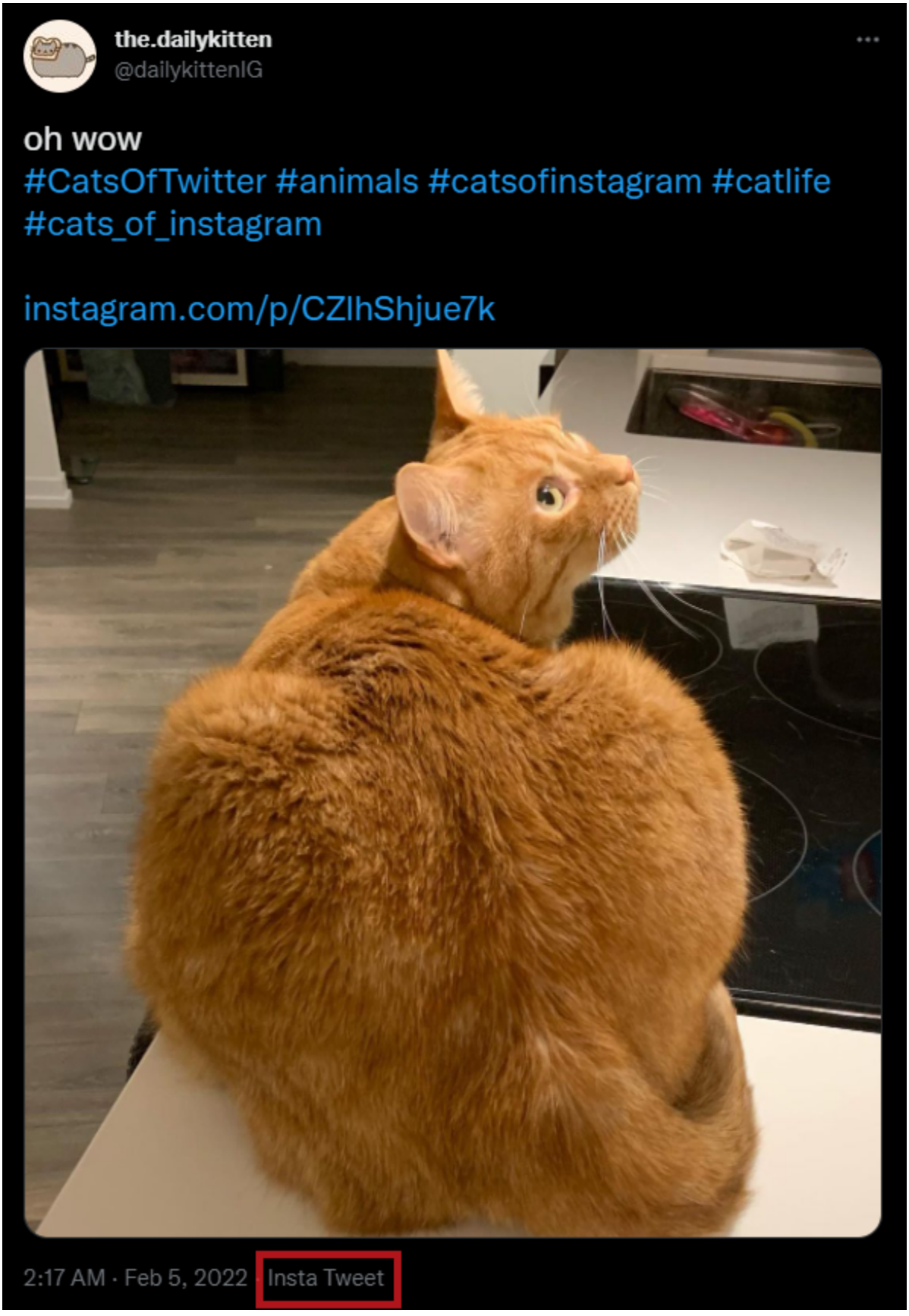
Ever tried sharing an Instagram post to Twitter, only to find out that all you tweeted was a link, and not the actual photo/video?



Humiliating

That could be literally anything. Nobody will click it.

`InstaTweet` **shares the** *actual* **content of the post. Not just a link to it.**


the.dailykitten
@dailykittenIG
oh wow
#CatsOfTwitter #animals #catsofinstagram #catlife
#cats_of_instagram
instagram.com/p/CZlhShjue7k
2:17 AM · Feb 5, 2022
Insta Tweet

With InstaTweet, you can rest easy knowing that, although nobody will click the link, they'll at least see what you posted.

1.2 What's InstaTweet?

InstaTweet is a customizable tool to automatically repost content from Instagram to Twitter.

Simply create a Profile, configure the *Mandatory Settings*, and add_pages() to repost from

```
from InstaTweet import Profile

# Create a new (local) Profile
>>> profile = Profile('myProfile')

# Configure the mandatory settings (at minimum)
>>> profile.twitter_keys = twitter_api_keys
>>> profile.session_id = '6011991A'

# Add at least one Instagram page (user/hashtag) to repost from
>>> profile.add_pages(['the.dailykitten', '#thedailykitten'])

 # Save the Profile [optional]
>>> profile.save()

Saved Local Profile myProfile
```

Once configured, the Profile can be used to initialize and start() InstaTweet:

```
from InstaTweet import InstaTweet

# Directly initialize with a Profile object
>>> insta_tweet = InstaTweet(profile)

# Or load a saved Profile by name
>>> insta_tweet = InstaTweet.load("myProfile")

# Run InstaTweet by calling start()
>>> insta_tweet.start()
```

From the Docs...

InstaTweet.**start**(*max_posts=12*)

InstaTweets all pages that have been added to the loaded Profile

The most recent posts from each page will be scraped, then compared to the scraped list in the PAGE_MAPPING to determine which are new.

Up to max_posts new posts from each page will then be downloaded and tweeted

---

Note: If InstaTweet fails to download_post() or send_tweet(), the PAGE_MAPPING won't be updated

- This ensures that failed repost attempts are retried in the next call to start()

If a save file for the Profile already `exists`, successful reposts will trigger a call to `save()`

Parameters
max_posts (`int`) – the maximum number of new posts to download and tweet per page

As `InstaTweet` runs, its progress will be logged to console:

```
Starting InstaTweet for Profile: myProfile
Checking posts from @the.dailykitten
...
Checking posts from #thedailykitten
...
Finished insta-tweeting for #thedailykitten
All pages have been insta-tweeted
```

1.3 Okay… But Why?

But Why?

InstaTweet has two main use cases:

- To automatically share your own Instagram posts to Twitter
- To automatically tweet new content from other Instagram users/hashtags

Regardless of your intention, InstaTweet will detect new posts from the pages you specify, download them, and repost them to Twitter.

...

1.3.1 Other Use Case: The `InstaClient`

The package's custom `InstaClient` can be used separately to scrape Instagram

```
from InstaTweet import InstaClient

>>> ig = InstaClient(session_id="kjfdn309wredsfl")

# Scrape Instagram user or hashtag
>>> user = ig.get_user('dailykittenig')
>>> hashtag = ig.get_hashtag('#dailykitten')
>>> print(user, hashtag, sep='\n')

Instagram User: @dailykittenig
Instagram Hashtag: #dailykitten

# Download most recent post
>>> post = user.posts[0]
>>> print(post)
```

(continues on next page)

(continued from previous page)

```
>>> ig.download_post(post)

Post 2981866202934977614 by @dailykittenig on 2022-11-29 01:44:37
Downloaded post https://www.instagram.com/p/Clht4NRrqRO by dailykittenig to C:\path\to\insta-
↪tweet\downloads\2981866202934977614.mp4
```

...

1.4 Installation

To install using pip:

```
pip install insta-tweet
```

Please note that `InstaTweet` requires `Python >= 3.8`

CHAPTER

TWO

GETTING STARTED

2.1 InstaTweet Profiles

InstaTweet uses the `Profile` class to help manage Twitter accounts, Instagram sessions, and page maps.

class InstaTweet.profile.**Profile**(*name='default'*, *local=True*, ***kwargs*)

The `Profile` is a configuration class used extensively throughout the package

It consists of a `page_map` and an associated collection of API/web scraping *settings*

. . .

About the Page Map

The `page_map` is a dict containing info about the pages added to a `Profile`

- It's used to help detect new posts and compose tweets on a per-page basis
- Entries are created when you `add_pages()`, which map the page to a `PAGE_MAPPING`
- The `PAGE_MAPPING` maintains lists of hashtags, scraped posts, and sent tweets
- The mapping is updated when you `add_hashtags()` and successfully `send_tweet()`

. . .

[Optional]

A unique, identifying `name` can optionally be assigned to the Profile, which may then be used to `save()` and `load()` its settings

The save location is determined by the value of `Profile.local` as follows:

- If `True`, saves are made locally to the `LOCAL_DIR` as .pickle files
- If `False`, saves are made remotely to a database as pickle bytes

See *Saving a Profile* for more information

. . .

2.2 Profile Settings

2.2.1 Mandatory Settings

- `session_id` — Instagram sessionid cookie, obtained by logging in on a desktop browser
- `twitter_keys` — Twitter API keys with v1.1 endpoint access

2.2.2 Mandatory Settings with Default Values

- `name` (="default") — the profile name; if non-default, it must be unique
- `local` (=True) — indicates if the profile should be saved locally (default) or on a remote database
- `user_agent=USER_AGENT` — user agent to use when making requests to Instagram; currently hardcoded

2.2.3 Entirely Optional Settings

- `proxy_key` — Environment variable to retrieve proxies from when making requests to Instagram/Twitter
- `page_map` — Fully formatted dictionary of IG pages mapped to their `PAGE_MAPPING`

2.3 Creating a Profile

Profile settings can be configured

1. By passing them as keyword arguments when initializing a `Profile`
2. By setting them directly as object attributes after the `Profile` object is created

```
from InstaTweet import Profile

# Initialize a profile with arguments
p = Profile('myProfile', session_id='6011991A')

# Initialize a profile with no arguments
q = Profile()
q.name = 'myProfile'
q.session_id = '6011991A'
```

All settings can be accessed via the `config` dict, which can be pretty printed using `view_config()`

```
# View and compare configuration settings
>>> q.view_config()
```

```
name : myProfile
local : True
session_id : 6011991A
twitter_keys : {'Consumer Key': 'string', 'Consumer Secret': 'string', 'Access Token':
↪'string', 'Token Secret': 'string'}
user_agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
```

(continues on next page)

(continued from previous page)

```
↪Gecko) Chrome/102.0.5005.63 Safari/537.36
proxy_key : None
page_map : {}
```

...

Validating Profile Settings

- Property setters validate data types for the *Mandatory Settings*
- Requirements aren't strictly enforced until `start()` is called, which will first `validate()` the profile settings

2.4 Populating the Page Map

2.4.1 The Page Map

The `page_map` allows a `Profile` to maintain a history of package-related activity for its added IG pages (users or hashtags).

Pages are mapped to their `PAGE_MAPPING`, which contains their associated lists of:

```
PAGE_MAPPING = {'hashtags': [], 'scraped': [], 'tweets': []}
```

- `hashtags` — the page's associated hashtag list (for use when composing tweets)
- `scraped` — the list of posts that have been scraped from the page (only the post id)
- `tweets` — the list of sent tweets containing media scraped from that page (limited data)

The mapping gets updated each time `InstaTweet` successfully scrapes and tweets a post from the page

2.4.2 Adding Pages

Use the `add_pages()` method to add one or more Instagram pages to a `Profile`'s `page_map`

```
from InstaTweet import Profile

# Add one page at a time
>>> p = Profile('myProfile')
>>> p.add_pages('the.dailykitten', send_tweet=True)

Added Instagram page @the.dailykitten to the page map

# Add multiple pages at once
>>> pages = ['dailykittenig','#thedailykitten']
>>> p.add_pages(pages)

Added Instagram page @dailykittenig to the page map
Added Instagram page #thedailykitten to the page map
```

The `get_page()` method can be used to retrieve the full `PAGE_MAPPING` of an added page

```
>> p.get_page('the.dailykitten')

{'hashtags': [], 'scraped': [-1], 'tweets': []}
```

2.4.3 Adding Hashtags

You can `add_hashtags()` for each page in the `page_map`

- They'll be chosen from at random when composing tweets based on one of their `posts`
- For more info, see `pick_hashtags()`, `build_tweet()` and `send_tweet()`

```
# Add a single hashtag for a specific page
>>> p.add_hashtags('dailykittenig', 'cats')

Added hashtags for dailykittenig

# Add multiple hashtags at once
>>> pages = ['the.dailykitten', '#thedailykitten']
>>> hashtags = ['kittygram', 'kittycat']

>>> for page in pages:
...     p.add_hashtags(page, hashtags)

Added hashtags for the.dailykitten
Added hashtags for #thedailykitten

>>> p.view_config()
```

```
name : myProfile
local : True
session_id : 6011991A
twitter_keys : {'Consumer Key': 'string', 'Consumer Secret': 'string', 'Access Token':
↪'string', 'Token Secret': 'string'}
user_agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/102.0.5005.63 Safari/537.36
proxy_key : None
page_map : {'the.dailykitten': {'hashtags': ['kittygram', 'kittycat'], 'scraped': [-1],
↪'tweets': []}, 'dailykittenig': {'hashtags': ['cats'], 'scraped': [], 'tweets': []}, '
↪#thedailykitten': {'hashtags': ['kittygram', 'kittycat'], 'scraped': [], 'tweets': []}}
```

2.4.4 Page Map Access Methods

Page Map Access Methods

The `Profile` has several methods that allow for easy access to the `page_map`

- `get_page()` provides access to a particular page's `PAGE_MAPPING`
- `get_scraped_from()` returns the list of posts scraped from a specified page
- `get_hashtags_for()` returns the list of hashtags to use in tweets for the specified page

- `get_tweets_for()` returns a list of tweets that use the specified page's scraped content

2.5 Saving a Profile

Saving a Profile

When you `save()` your `Profile`, the current or specified `name` will be used to create or update a save file in the location specified by `local`

From the Docs...

`Profile.`**`save`**(*name=None*, *alert=True*)

Pickles and saves the `Profile` using the specified or currently set name.

Parameters

- **`name`** (`Optional[str]`) – name to save the `Profile` under; replaces the current `name`
- **`alert`** (`bool`) – set to `True` to print a message upon successful save

Return type

bool

`InstaTweet.profile.Profile.`**`local =True`**

Indicates if saves should be made locally (`True`) or on a remote database (`False`)

- Local saves are made to the `LOCAL_DIR`, as pickle files
- Remote saves are made to a database (via the `db` module) as pickle bytes

Important!!

You MUST configure the `DATABASE_URL` environment variable to save/load remotely

InstaTweet uses `SQLAlchemy` to create a `DBConnection`

- Any `SQLAlchemy`-supported database is therefore also supported by `InstaTweet`
- See the `db` module for more information

2.5.1 Example: Save a Profile

Note: You can specify a new `name` for the profile in the call to `save()`

```
from InstaTweet import Profile

>>> p = Profile('myProfile')
>>> p.save()
```

(continues on next page)

(continued from previous page)

```
Saved Local Profile myProfile

>>> p.save('aProfile')
>>> print(p.name)

Saved Local Profile aProfile
aProfile
```

Profile names must be unique - you cannot save or create a profile if a `profile_exists()` with that name already

```
>>> q = Profile('myProfile')

FileExistsError: Local save file already exists for profile named "myProfile"
Please choose another name, load the profile, or delete the file.
```

2.6 Running a Profile

Once a `Profile` is configured, it can be used to initialize and `start()` an `InstaTweet` object

```
from InstaTweet import InstaTweet, Profile

# Load an existing saved or unsaved profile into InstaTweet
>>> profile = Profile.load("myProfile")
>>> insta_tweet = InstaTweet(profile)

# Or directly InstaTweet.load() the settings in by Profile name
>>> insta_tweet = InstaTweet.load(profile_name="myProfile")

# Then run InstaTweet by calling start()
>>> insta_tweet.start()
```

From the Docs...

`InstaTweet.`**`start`**(*max_posts=12*)

InstaTweets all pages that have been added to the loaded `Profile`

The most recent posts from each page will be scraped, then compared to the `scraped` list in the `PAGE_MAPPING` to determine which are new.

Up to `max_posts` new posts from each page will then be downloaded and tweeted

---

Note: If `InstaTweet` fails to `download_post()` or `send_tweet()`, the `PAGE_MAPPING` won't be updated

- This ensures that failed repost attempts are retried in the next call to `start()`

If a save file for the Profile already `exists`, successful reposts will trigger a call to `save()`

Parameters

`max_posts` (`int`) – the maximum number of new posts to download and tweet per page

As InstaTweet runs, its progress will be logged to console:

```
Starting InstaTweet for Profile: myProfile
Checking posts from @the.dailykitten
...
Checking posts from #thedailykitten
...
Finished insta-tweeting for #thedailykitten
All pages have been insta-tweeted
```

CHAPTER

THREE

SCHEDULE INSTATWEET

You can easily schedule InstaTweet using the provided scheduler script

```
#scheduler.py
from InstaTweet import InstaTweet

PROFILES = ['aProfile', 'myProfile']
LOCAL = True


def run(profile_name: str, local: bool = LOCAL):
    """Loads and InstaTweets a profile

    :param profile_name: the name of the :class:`~.Profile`
    :param local: if the profile is saved locally or in a SQLAlchemy supported database
    """
    insta_tweet = InstaTweet.load(profile_name, local=local)
    insta_tweet.start()


if __name__ == '__main__':
    for profile in PROFILES:
        run(profile, local=LOCAL)
```

CHAPTER

FOUR

INSTATWEET PACKAGE

Below, you'll find the documentation for each class/module in the `InstaTweet` package.

4.1 The `InstaTweet` class

Not to be confused with the package, the `InstaTweet` class is the main point of interaction for a user of the package

- It uses a fully configured `Profile` to initialize an `InstaClient` and `TweetClient`, which are used to scrape and tweet posts when `start()` is called

...

class InstaTweet.instatweet.**InstaTweet**(*profile*)

Bases: `object`

Uses the settings from a Profile to do the actual InstaTweeting

You might be wondering, what's InstaTweeting? According to TDK Dictionary:

InstaTweet (*verb*):

To load a `Profile` scrape `posts` from its Instagram pages `download_post()` & `send_tweet()` for any new content update the `page_map` `save()` the profile if it `exists`

Example Sentence

Oh, you lost 700 Twitter followers after you shared your IG post? Well maybe if people actually saw the picture and not just the caption your tweet would've been less creepy. You should've InstaTweeted it.

__init__(*profile*)

Initializes `InstaTweet` using a fully configured `Profile`

The `Profile` will be used to initialize an `InstaClient` and `TweetClient`

Note: InstaTweet won't `validate()` the Profile settings until you call `start()`

Parameters
profile (Profile) – the Profile to use for InstaTweeting

classmethod load(*profile_name*, *local=True*)

Loads a profile by name

Parameters

- **profile_name** (str) – name of the Profile to load
- **local** (bool) – whether the profile is saved locally (default) or remotely on a database

Return type
InstaTweet

get_proxies()

Retrieve proxies using the loaded Profile's proxy_key

Return type
Optional[*Dict*]

get_insta_client()

Initializes an InstaClient using the loaded Profile settings

Return type
InstaClient

get_tweet_client()

Initializes an TweetClient using the loaded Profile settings

Return type
TweetClient

start(*max_posts=12*)

InstaTweets all pages that have been added to the loaded Profile

The most recent posts from each page will be scraped, then compared to the scraped list in the PAGE_MAPPING to determine which are new.

Up to max_posts new posts from each page will then be downloaded and tweeted

---

Note: If InstaTweet fails to download_post() or send_tweet(), the PAGE_MAPPING won't be updated

- This ensures that failed repost attempts are retried in the next call to start()

If a save file for the Profile already exists, successful reposts will trigger a call to save()

Parameters
max_posts (int) – the maximum number of new posts to download and tweet per page

get_new_posts(*insta_page*)

Scrapes recent posts from an Instagram page and returns all posts that haven't been tweeted yet

NOTE: If a page's scraped list is empty, no posts will be returned.

Instead, the page is "initialized" as follows:

- The `scraped` list will be populated with the ID's from the most recent posts
- These IDs are then used in future method calls to determine which posts to tweet

Parameters
insta_page (`str`) – the Instagram page to scrape posts from

Returns
a list of posts that haven't been tweeted yet, or nothing at all (if page is only initialized)

Return type
Optional[*List*[InstaPost]]

4.2 The `Profile` class

class InstaTweet.profile.**Profile**(*name='default'*, *local=True*, ***kwargs*)

Bases: `object`

The `Profile` is a configuration class used extensively throughout the package

It consists of a `page_map` and an associated collection of API/web scraping *settings*

. . .

About the Page Map

The `page_map` is a dict containing info about the pages added to a `Profile`

- It's used to help detect new posts and compose tweets on a per-page basis
- Entries are created when you `add_pages()`, which map the page to a `PAGE_MAPPING`
- The `PAGE_MAPPING` maintains lists of hashtags, scraped posts, and sent tweets
- The mapping is updated when you `add_hashtags()` and successfully `send_tweet()`

. . .

[Optional]

A unique, identifying `name` can optionally be assigned to the Profile, which may then be used to `save()` and `load()` its settings

The save location is determined by the value of `Profile.local` as follows:

- If `True`, saves are made locally to the `LOCAL_DIR` as .pickle files
- If `False`, saves are made remotely to a database as pickle bytes

See *Saving a Profile* for more information

. . .

PAGE_MAPPING: Dict = {'hashtags': [], 'scraped': [], 'tweets': []}

Template for an entry in the `page_map`

LOCAL_DIR: str = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/instatweet/envs/latest/lib/python3.8/site-packages/profiles')

Directory where local profiles are saved

__init__(*name='default'*, *local=True*, ***kwargs*)

Create a new Profile

Parameters

- **name** (str) – unique profile name
- **local** (bool) – indicates if profile is being saved locally or on a remote database
- **kwargs** – see below

Keyword Arguments

- ***session_id*** **(str)**
  Instagram sessionid cookie, obtained by logging in through browser
- ***twitter_keys*** **(dict)**
  Twitter API Keys with v1.1 endpoint access (see DEFAULT_KEYS for a template)
- ***user_agent*** **(str) – Optional**
  The user agent to use for requests
- ***proxy_key*** **(str) – Optional**
  Environment variable to retrieve proxies from

user_agent: str

The user agent to use for requests

proxy_key: str

Environment variable to retrieve proxies from

page_map: Dict[str, Dict]

Mapping of added Instagram pages and their PAGE_MAPPING

classmethod load(*name*, *local=True*)

Loads an existing profile from a locally saved pickle file or remotely stored pickle bytes

Parameters

- **name** (str) – the name of the Profile to load
- **local** (bool) – whether the profile is saved locally (default, True) or remotely on a database

Return type
Profile

classmethod from_json(*json_str*)

Creates a profile from a JSON formatted string of config settings

Return type
Profile

classmethod from_dict(*d*)

Creates a profile from a dictionary of config settings

Return type
Profile

static profile_exists(*name*, *local=True*)

Checks locally/remotely to see if a `Profile` with the specified name has an existing save file

Whenever the `name` is changed, its property setter calls this method to ensure you don't accidentally overwrite a save that already `exists`

Parameters

- **name** (`str`) – the name of the `Profile` to check for
- **local** (`bool`) – the location (local/remote) to check for an existing save

Return type
bool

static get_local_path(*name*)

Returns filepath of where a local profile would be saved

Return type
str

add_pages(*pages*, *send_tweet=False*)

Add Instagram page(s) to the `page_map` for subsequent monitoring

- An Instagram profile can be added as `"@username"` or `"username"`
- A hashtag must be added as `"#hashtag"`

Note: By default, newly added pages won't have their posts tweeted the first time they're scraped

- The IDs of the most recent posts are stored in the `scraped` list
- Any new posts from that point forward will be tweeted

You can scrape AND tweet posts on the first run by setting `send_tweet=True`

Parameters

- **pages** (`Iterable`) – Instagram pages to automatically scrape and tweet content from
- **send_tweet** (`bool`) – choose if tweets should be sent on the first scrape, or only for new posts going forward

add_hashtags(*page*, *hashtags*)

Add hashtag(s) to a page in the `page_map`, which will be randomly chosen from when composing Tweets

Parameters

- **page** (`str`) – the page in the page map to add hashtags to
- **hashtags** (`Iterable`) – hashtags to choose from and include in any Tweets where content comes from this page

save(*name=None*, *alert=True*)

Pickles and saves the `Profile` using the specified or currently set name.

Parameters

- **name** (`Optional[str]`) – name to save the `Profile` under; replaces the current `name`

- **alert** (bool) – set to True to print a message upon successful save

Return type
bool

validate()
Checks to see if the Profile is fully configured for InstaTweeting

Raises
ValueError – if the session_id, twitter_keys, or page_map are invalid

to_pickle()
Serializes profile to a pickled byte string

Return type
bytes

to_json()
Serializes profile to a JSON formatted string

Return type
str

to_dict()
Serializes profile to a dict

Return type
dict

view_config()
Prints the config dict to make it legible

property config: dict
Returns a dictionary containing important configuration settings

property exists: bool
Returns True if a local save file or database record exists for the currently set profile name

property is_default: bool
Check if profile name is set or not

property profile_path: str
If local is True, returns the file path for where this profile would be/is saved

get_page(*page*)
Returns the specified page's dict entry in the page_map

Return type
dict

get_scraped_from(*page*)
Returns a list of posts that have been scraped from the specified page

Return type
list

get_tweets_for(*page*)
Returns a list of tweets that use the specified page's scraped content

Return type
list

get_hashtags_for(*page*)

Returns the hashtag list for the specified page

Return type
list

property local: bool

Indicates if saves should be made locally (`True`) or on a remote database (`False`)

property name: str

A name for the Profile

The `name` is used differently depending on the value of `local`

- `local==True`: the name determines the `profile_path` (path where it would save to)
- `local==False`: the name is used as the primary key in the `Profiles` database table

...

Profile Names Must Be Unique

When you set or change the `name`, a property setter will make sure no `profile_exists()` with that name before actually updating it

- This ensures that you don't accidentally overwrite a different Profile's save data

---

...

Raises

- **FileExistsError** – if `local` ==`True` and a save is found in the `LOCAL_DIR`
- **ResourceWarning** – if `local` ==`False` and a database row is found by `query_profile()`

property session_id: str

Instagram `sessionid` cookie, obtained by logging in through browser

property twitter_keys: Dict

Twitter API Keys with v1.1 endpoint access (see `DEFAULT_KEYS` for a template)

4.3 The db module

The `db` module contains the `DBConnection` class and the `Profiles` database table

---

The Database Table

In the `Profiles` database table each row corresponds to a unique `Profile`

The table only has two fields per row:

- `name`: primary key for lookups/insertions
- `config`: stores the Profile as pickle bytes via `to_pickle()`

How is profile data saved to the database?

When a `Profile` calls `save()` and has `local = False`, it will `connect()` to the database specified by the `DABATASE_URL` environment variable and use it to `query_profile()` settings

- If the `profile_exists()` in the database already, its `config` data will be updated
- Otherwise, the `DBConnection` will `save_profile()` data in a new table row

Important!!

You MUST configure the `DATABASE_URL` environment variable to save/load remotely

- InstaTweet uses `SQLAlchemy` to create a `DBConnection` – any db it supports is compatible
- See the `db` module for more information

One Last Thing!

The `DBConnection` is meant to be used as a context manager

```
with DBConnection() as db:
    # Do Something
```

- A `SESSION` is created/destroyed when saving, loading, and InstaTweeting a `Profile`

If you don't want that, here's instructions on *Persisting The DBConnection*

...

`InstaTweet.db.`**`DATABASE_URL`**

The Database URL to use, obtained from the `DATABASE_URL` environment variable

...

`class` `InstaTweet.db.`**`Profiles`**(***kwargs*)

Database table used for storing `Profile` settings

The table currently has only 2 fields, for the `name` and pickle bytes of the profile

`name`

The `Profile` name

`config`

The pickle bytes from `Profile.to_pickle()`

...

`class` `InstaTweet.db.`**`DBConnection`**

Database Connection class with context management ooh wow

Uses `SQLAlchemy` to connect and interact with the database specified by `DATABASE_URL` environment variable

Sample Usage:

```
def poop_check():
    with DBConnection() as db:
        if db.query_profile(name="POOP").first():
            raise FileExistsError('DELETE THIS NEPHEW......')
```

SESSION

The currently active session; closed on object exit

Type
scoped_session

ENGINE

The engine for the currently set DATABASE_URL; reused after first connection

Type
Engine

static connect()

Creates a scoped_session and assigns it to DBConnection.SESSION

query_profile(*name*)

Queries the database for a Profile by its name

Parameters
name (str) – the profile name (ie. the Profile.name)

Returns
the Query NOT the Profile

Return type
Query

load_profile(*name*)

Loads a profile from the database by name

Parameters
name (str) – the profile name (ie. the Profile.name)

Raises
LookupError – if the database has no profile saved with the specified name

Return type
Profile

save_profile(*profile*, *alert=True*)

Saves a Profile to the database by either updating an existing row or inserting a new one

Parameters

- **profile** (Profile) – the Profile to save
- **alert** (bool) – if True, will print a message upon successfully saving

Return type
bool

delete_profile(*name*, *alert=True*)

Deletes a Profile from the database by name

Parameters

- **name** (str) – the profile name (ie. the Profile.name)
- **alert** (bool) – if True, will print a message upon successfully deleting

Return type
bool

4.4 The TweetClient class

class InstaTweet.tweetclient.**TweetClient**(*profile*, *proxies=None*)

Bases: object

MAX_HASHTAGS = 5

DEFAULT_KEYS = {'Access Token': 'string', 'Consumer Key': 'string', 'Consumer Secret': 'string', 'Token Secret': 'string'}

__init__(*profile*, *proxies=None*)

Initialize TweetClient using a Profile

Basically just a wrapper for tweepy. It uses the settings of a profile to initialize the API and send tweets

Parameters

- **profile** (Profile) – the profile to use when initializing a tweepy.API object
- **proxies** (Optional[dict]) – optional proxies to use when making API requests

get_api()

Initializes a API object using the API keys of the loaded Profile

Return type
API

static get_oauth(*api_keys*)

Initializes and returns an OAuth1UserHandler object from tweepy using the specified API keys

Parameters
api_keys (dict) – Twitter developer API keys with v1.1 endpoint access

Return type
OAuth1UserHandler

send_tweet(*post*, *hashtags=None*)

Composes and sends a Tweet using an already-downloaded Instagram post

---

How Tweets are Sent

The InstaPost.filepath – set by download_post() – is used as the media source

The body of the tweet is then generated by build_tweet()

Parameters

- **post** (InstaPost) – the post to tweet

- **hashtags** (Optional[list[str]]) – a list of hashtags to randomly chose from and include in the tweet

Return type
bool

upload_media(*post*)

Uploads the media from an already-downloaded Instagram post to Twitter

---

Note: If the post is a carousel, only the first 4 photos/videos will be uploaded

---

Parameters
post (InstaPost) – the Instagram post to use as the media source

Returns
the list of uploaded media ids (if API upload was successful) or False

Return type
Union[list, bool]

build_tweet(*post*, *hashtags=None*)

Uses an InstaPost to build the body text of a tweet

How Tweets are Composed

- The caption is used as a starting point
- If you add_hashtags() for the page, it will randomly pick_hashtags() to include
- Lastly, the post's permalink is added to the end

Example:

```
>> post = instatweet.insta.get_user("dailykittenig").posts[0]
>> tweet = instatweet.twitter.build_tweet(post)
>> print(tweet)

carousel support yuh
#kitten #kittycat #catlover #petstagram #animals

https://www.instagram.com/p/Cjl3UWBOd8k
```

Parameters

- **post** (InstaPost) – the post being tweeted
- **hashtags** (Optional[list[str]]) – optional list of hashtags to randomly pick from and include

Returns
the text to use for the tweet

Return type
str

static pick_hashtags(*hashtags*)

Randomly picks hashtags from the provided list and returns them as a single string

The number of hashtags chosen will either be 1 less than the length of the list (to avoid using the same tags in every tweet), or the value of MAX_HASHTAGS, whichever is smaller

Parameters

hashtags (list[str]) – a list of hashtags to randomly choose from

Example

```
from InstaTweet import TweetClient

>> TweetClient.pick_hashtags(['cat','dog','woof'])
"#woof #cat\n"
```

Return type

str

Note: A newline is added to help with formatting & character counting in build_tweet()

4.5 The InstaClient class

InstaTweet.instaclient.**USER_AGENT**

Hardcoded user agent proven to work with the get_user() endpoint

Version Added

v2.0.0b13

InstaClient.**DOWNLOAD_DIR**

[*Optional*] – Directory to temporarily download media to

class InstaTweet.instaclient.**InstaClient**(*session_id*, *user_agent=USER_AGENT*, *proxies=None*)

Bases: object

Minimalistic class for scraping/downloading Instagram user/media data

__init__(*session_id*, *user_agent=USER_AGENT*, *proxies=None*)

Initialize an InstaClient with an Instagram sessionid cookie (at minimum)

Note: As of v2.0.0b13, the endpoint used by get_user() seems to require a specific USER_AGENT You can override the hardcoded one if you'd like, but you'll likely get a "useragent mismatch" response

Parameters

- **session_id** (str) – valid Instagram sessionid cookie from a browser
- **user_agent** (str) – user agent to use in requests made by the class
- **proxies** (Optional[Dict]) – proxies to use in requests made by the class

request(*url*)

Sends a request using the `cookies`, `headers`, and `proxies`

Parameters
url (`str`) – the Instagram URL to send the request to

Return type
Response

scrape(*page*)

Scrapes an Instagram page and wraps the response data

Parameters
page (`str`) – an Instagram hashtag (prefixed with #) or username

Returns
an `InstaUser` or `Hashtag`

Return type
InstaPage

get_hashtag(*tag*, *max_id=''*)

Scrapes an Instagram hashtag and wraps the response with `Hashtag`

Parameters

- **tag** (`str`) – the hashtag to scrape (with or without a #)
- **max_id** (`str`) – the end cursor

Return type
Hashtag

get_user(*username*)

Scrapes an Instagram user's profile and wraps the response with `InstaUser`

Parameters
username (`str`) – the username of the IG user to scrape

Return type
InstaUser

get_post(*shortcode*)

Scrapes an Instagram post by shortcode or URL

Parameters
shortcode (`str`) – the shortcode or URL of the post

Return type
Optional[InstaPost]

get_username(*user_id*)

Retrieves the Instagram username for the user with the provided `user_id`

Tip: Use this with `get_user()` to scrape by `user_id`:

```
>> user_id = 51276430399
>> username = insta.get_username(user_id)
>> user = insta.get_user(username)
>> print(user.posts[0])
```

Post 2981866202934977614 by @dailykittenig on 2022-11-29 01:44:37

Parameters
user_id (Union[int, str]) – the id of the Instagram user to retrieve the username of

Return type
str

download_post(*post*, *filepath=None*)

Downloads the media from an Instagram post

Parameters

- **post** (InstaPost) – the InstaPost of the post to download
- **filepath** (Optional[str]) – the path to save the downloaded media; if None, saves to the DOWNLOAD_DIR

Return type
bool

property headers: Dict

Headers to use in request()

property cookies: Dict

Cookies to use in request()

4.6 The InstaPage module

class InstaTweet.instapage.**InstaPage**(*data*, *client=None*)

Bases: ABC

Abstract wrapper class for wrapping API responses from Instagram pages

__init__(*data*, *client=None*)

Initialize an InstaPage

Used to wrap responses from endpoints that contain Instagram post data, like Instagram user profiles and Instagram hashtag searches

Parameters

- **data** (Dict) – the API response JSON to use as source data
- **client** (Optional[InstaClient]) – the InstaClient to use; required for get_more_posts()

abstract property name: str

Name of the Instagram page

abstract property page_data: Dict

Data about the Instagram page itself

abstract property media_data: Dict

Data about posts on the Instagram page

property id: int

ID of the Instagram page

property posts: List[InstaPost]

Posts that have been scraped from the Instagram page

To retrieve the next page of posts, call `get_more_posts()`

Returns

the page's posts as `InstaPost` objects

get_more_posts()

Requests the next page of posts from the `InstaPage`

If the page `has_more_posts`, they'll be added to the `posts` list

Returns

`True` if the request was successful, otherwise `False`

Return type

bool

property has_more_posts: bool

Returns `True` if more posts can be scraped using `get_more_posts()`

property end_cursor: str

Cursor used in request by `get_more_posts()`

property media_page_info: Dict

class InstaTweet.instapage.**InstaUser**(*data*, *client=None*)

Bases: `InstaPage`

API response wrapper for an Instagram user's profile

__init__(*data*, *client=None*)

Initialize an `InstaUser`

Parameters

- **data** (`Dict`) – the API response from `get_user()`
- **client** (`Optional[InstaClient]`) – the `InstaClient` to use

property name: str

Name of the Instagram page

property page_data: Dict

Data about the Instagram page itself

property media_data: Dict

Data about posts on the Instagram page

class InstaTweet.instapage.**Hashtag**(*data*, *client=None*)

Bases: `InstaPage`

API response wrapper for an Instagram hashtag

__init__(*data*, *client=None*)

Initialize a Hashtag

Parameters

- **data** (Dict) – the API response from get_hashtag()
- **client** (Optional[InstaClient]) – the InstaClient to use

property name: str

Name of the Instagram page

property page_data: Dict

Data about the Instagram page itself

property media_data: Dict

Data about posts on the Instagram page

property top_posts: List[InstaPost]

property top_media_data: Dict

4.7 The InstaPost class

class InstaTweet.instapost.**InstaPost**(*data*, *client=None*)

Bases: object

Minimalistic API response wrapper for an Instagram post

__init__(*data*, *client=None*)

Initialize an InstaPost

Parameters

data (dict) – the JSON response data of a single Instagram post, found within the user_data

json

Source data from API response

id

The post id

filepath: str

Path of downloaded media, set by download_post()

tweet_data: dict

Limited data from a successful tweet based off this post, set by send_tweet()

property children: List[InstaPost]

If the post is a carousel, returns a list of child InstaPost's

property permalink: str

property shortcode: str

property caption: str

property likes: Optional[int]

property media_url: str

The direct URL to the actual post content

Returns
the `video_url` if the post is a video, otherwise the `thumbnail_url`

property thumbnail_url: str

property is_downloaded: bool

Checks the `filepath` to see if the post has been downloaded yet

property is_carousel: bool

property filename: str

Concatenates `id` + `filetype` to create the default filename, for use when saving the post

For Example:

```
>> print(post.filename)
"2868062811604347946.mp4"
```

property filetype: str

Filetype of the post, based on the value of `is_video`

property owner: Dict

property timestamp: Union[datetime, str]

add_tweet_data(*tweet*)

Used by `TweetClient` to add minimal tweet data after the post has been tweeted

Parameters
tweet (`Status`) – a `Status` object from a successfully sent tweet

Return type
bool

4.8 The `utils` module

This module contains a few helper functions that are used throughout the package

`InstaTweet.utils.`**get_agents**()

Scrapes a list of user agents. Returns a default list if the scrape fails.

Note: Deprecated since 2.0.0b13, but might be useful when new endpoint gets patched

---

Return type
list

InstaTweet.utils.**get_agent**(*index=0*)

Returns a single user agent string from the specified index of the AGENTS list

Note: Deprecated since 2.0.0b13, but might be useful when new endpoint gets patched

Return type
str

InstaTweet.utils.**get_proxies**(*env_key*)

Retrieve proxies from an environment variable

Return type
Optional[dict]

InstaTweet.utils.**get_root**()

Return type
Path

InstaTweet.utils.**get_filepath**(*filename*, *filetype='txt'*)

Return type
str

Just want to get started?

If you don't care about the details and just want to get this running... I get you.

You'll only need to be familiar with

- The `Profile`, which is used to configure all *settings*
- The `InstaTweet` class, which is used to `start()` the "main script"
- The `db` module, but only if you plan to save data remotely

Otherwise, the other classes are pretty self explanatory

Tip:

- `InstaClient` sends requests to Instagram
- `InstaPost` and `InstaUser` wrap responses from Instagram
- `TweetClient` wraps the `tweepy.API` to `send_tweet()` based off an `InstaPost`
- The `db` module contains the `Profiles` database table and the `DBConnection` class

CHAPTER

FIVE

INSTATWEET SNIPPETS

5.1 About the Page Map

About the Page Map

The `page_map` is a dict containing info about the pages added to a `Profile`

- It's used to help detect new posts and compose tweets on a per-page basis
- Entries are created when you `add_pages()`, which map the page to a `PAGE_MAPPING`
- The `PAGE_MAPPING` maintains lists of hashtags, scraped posts, and sent tweets
- The mapping is updated when you `add_hashtags()` and successfully `send_tweet()`

You can access entries in the `page_map` as follows:

- `get_page()` allows you to retrieve a full entry by page name
- `get_hashtags_for()`, `get_scraped_from()`, `get_tweets_for()` provide access to lists

5.2 Persisting The DBConnection

You can assign a `DBConnection()` to a variable if you want a persistent connection

Here's how:

```
from InstaTweet import DBConnection

# When __enter__ is called for the first time, the engine is set
>>> with DBConnection() as db:
...     pass

# Since the database URL is constant,
#__exit__() doesn't remove the ENGINE class var
>>> print(db.ENGINE)
Engine(postgresql://...)

# The SESSION class var is cleared upon __exit__ though
>>> print(db.SESSION)
None
```

To connect to the database and create a new session, call `connect()`

It will persist until you somehow trigger a call to `__exit__()`

```
# Using the DBConnection object from above
# Call connect() to create a new connection
>>> db.connect()

# Now it can be used like a regular object, and the
# connection will persist until you trigger a call to __exit__()
>>> profile = db.load_profile('myProfile')
>>> profile.view_config()
```

Output:

```
name : myProfile
local : True
session_id :
twitter_keys : {'Consumer Key': 'string', 'Consumer Secret': 'string', 'Access Token':
↪'string', 'Token Secret': 'string'}
user_agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/102.0.5005.63 Safari/537.36
proxy_key : None
user_map : {}
```

yea

5.3 Running a Profile

Once a `Profile` is configured, it can be used to initialize and `start()` an `InstaTweet` object

```
from InstaTweet import InstaTweet, Profile

# Load an existing saved or unsaved profile into InstaTweet
>>> profile = Profile.load("myProfile")
>>> insta_tweet = InstaTweet(profile)

# Or directly InstaTweet.load() the settings in by Profile name
>>> insta_tweet = InstaTweet.load(profile_name="myProfile")

# Then run InstaTweet by calling start()
>>> insta_tweet.start()
```

From the Docs...

`InstaTweet.`**`start`**(*max_posts=12*)

InstaTweets all pages that have been added to the loaded `Profile`

The most recent posts from each page will be scraped, then compared to the `scraped` list in the `PAGE_MAPPING` to determine which are new.

Up to `max_posts` new posts from each page will then be downloaded and tweeted

---

Note: If `InstaTweet` fails to `download_post()` or `send_tweet()`, the `PAGE_MAPPING` won't be updated

- This ensures that failed repost attempts are retried in the next call to `start()`

If a save file for the Profile already `exists`, successful reposts will trigger a call to `save()`

Parameters
max_posts (`int`) – the maximum number of new posts to download and tweet per page

As `InstaTweet` runs, its progress will be logged to console:

```
Starting InstaTweet for Profile: myProfile
Checking posts from @the.dailykitten
...
Checking posts from #thedailykitten
...
Finished insta-tweeting for #thedailykitten
All pages have been insta-tweeted
```

5.4 Saving a Profile

Saving a Profile

When you `save()` your `Profile`, the current or specified `name` will be used to create or update a save file in the location specified by `local`

From the Docs...

`Profile.`**save**(*name=None*, *alert=True*)
Pickles and saves the `Profile` using the specified or currently set name.

Parameters
- **name** (`Optional[str]`) – name to save the `Profile` under; replaces the current `name`
- **alert** (`bool`) – set to `True` to print a message upon successful save

Return type
bool

`InstaTweet.profile.Profile.`**local =True**
Indicates if saves should be made locally (`True`) or on a remote database (`False`)

- Local saves are made to the `LOCAL_DIR`, as pickle files
- Remote saves are made to a database (via the `db` module) as pickle bytes

Important!!

You MUST configure the `DATABASE_URL` environment variable to save/load remotely

InstaTweet uses `SQLAlchemy` to create a `DBConnection`

- Any `SQLAlchemy`-supported database is therefore also supported by `InstaTweet`

- See the db module for more information

5.4.1 Example: Save a Profile

Note: You can specify a new name for the profile in the call to save()

```
from InstaTweet import Profile

>>> p = Profile('myProfile')
>>> p.save()

Saved Local Profile myProfile

>>> p.save('aProfile')
>>> print(p.name)

Saved Local Profile aProfile
aProfile
```

Profile names must be unique - you cannot save or create a profile if a profile_exists() with that name already

```
>>> q = Profile('myProfile')

FileExistsError: Local save file already exists for profile named "myProfile"
Please choose another name, load the profile, or delete the file.
```

5.5 Schedule InstaTweet

You can easily schedule InstaTweet using the provided scheduler script

```
#scheduler.py
from InstaTweet import InstaTweet

PROFILES = ['aProfile', 'myProfile']
LOCAL = True


def run(profile_name: str, local: bool = LOCAL):
    """Loads and InstaTweets a profile

    :param profile_name: the name of the :class:`~.Profile`
    :param local: if the profile is saved locally or in a SQLAlchemy supported database
    """
    insta_tweet = InstaTweet.load(profile_name, local=local)
    insta_tweet.start()
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    for profile in PROFILES:
        run(profile, local=LOCAL)
```

5.6 Other Use Case: The `InstaClient`

The package's custom `InstaClient` can be used separately to scrape Instagram

```
from InstaTweet import InstaClient

>>> ig = InstaClient(session_id="kjfdn309wredsfl")

# Scrape Instagram user or hashtag
>>> user = ig.get_user('dailykittenig')
>>> hashtag = ig.get_hashtag('#dailykitten')
>>> print(user, hashtag, sep='\n')

Instagram User: @dailykittenig
Instagram Hashtag: #dailykitten

# Download most recent post
>>> post = user.posts[0]
>>> print(post)
>>> ig.download_post(post)

Post 2981866202934977614 by @dailykittenig on 2022-11-29 01:44:37
Downloaded post https://www.instagram.com/p/Clht4NRrqRO by dailykittenig to C:\path\to\insta-
↪tweet\downloads\2981866202934977614.mp4
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- *Full Table of Contents*

PYTHON MODULE INDEX

i

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V