Persine

Release 0.1.4

Jonathan Soma

Jan 14, 2021

CONTENTS

Release v0.1.4. (*Installation*)

Persine is an automated tool to study and reverse-engineer algorithmic recommendation systems, like YouTube videos and Amazon products. It has a simple interface and encourages reproducible results.

Persine is dead simple to use!

```
from persine import PersonaEngine

engine = PersonaEngine()

with engine.persona() as persona:
   persona.run_batch([
      "https://www.youtube.com/watch?v=hZw23sWlyG0",
      "youtube:next_up#3",
      "https://www.youtube.com/watch?v=hZw23sWlyG0"
   ])
   persona.history.to_csv("history.csv")
   persona.recommendations.to_csv("recs.csv")
```

In this example, we visit a YouTube video, click the "next up" video three times, and then revisit the original video. We then save the results for later analysis.

See Persine in action on Google Colab.

CHAPTER

ONE

THE USER GUIDE

1.1 Installation

1.1.1 Installing Persine

Installation of Persine isn't too hard, it can be installed using pip:

```
pip install persine
```

In order to do its job, Persine will automatically install a lot of dependencies! These include:

- Selenium to control the browser
- BeautifulSoup for browsing/scraping
- pandas for data analysis
- pillow for processing screenshots

After you install Persine, you aren't done yet: because Persine controls a browser, you need to install both a **browser** as well as the **software that connects Python to the browser**.

1.1.2 Installing Chrome

Persine uses Google Chrome to drive around the internet and pretend to be a user. I would love to switch to Firefox but it has its own problems.

You'll want to make sure it's in a normal place (for example, not just living in your "Downloads" folder).

1.1.3 Installing ChromeDriver

You will need to install ChromeDriver to allow Selenium to control Chrome. You can read the ChromeDriver getting started page but I've also included installing instructions below.

> Note that every time you update Chrome you'll need to update ChromeDriver.

Installing ChromeDriver on OS X

The easiest way to install ChromeDriver on OS X is using homebrew:

```
brew install --cask chromedriver
```

Alternatively, you can follow these steps:

1. Visit the ChromeDriver website
2. Click the "latest stable release" link
3. Download **chromedriver_mac64.zip**
4. Unzip it, revealing a file called **chromedriver** (no extension)
5. Move this file into your PATH. I typically put it in **/usr/local/bin**.

Installing ChromeDriver on Windows

Follow the following steps to install ChromeDriver on Windows:

1. Visit the ChromeDriver website
2. Click the "latest stable release" link
3. Download **chromedriver_win32.zip**
4. Unzip it, revealing a file called **chromedriver.exe** (no extension)
5. Move this file into your PATH. In the spirit of anarchy I just put it in **C:Windows**.

Installing ChromeDriver on Debian/Ubuntu

It's the easiest of them all:

```
apt install chromium-chromedriver
```

1.2 Quickstart

Make sure Persine and its dependencies are *installed* before you get started!

1.2.1 Introduction

In this example, we visit a specific YouTube video and click the next-up video three times to see where it leads us. We then save the results for later analysis.

```
from persine import PersonaEngine

engine = PersonaEngine()

with engine.persona() as persona:
    persona.run('https://www.youtube.com/watch?v=hZw23sWlyG0')
    persona.run('youtube:next_up#3')
    persona.history.to_csv('history.csv')
    persona.recommendations.to_csv('recs.csv')
```

1.2.2 Using personas

Persine is built around an **engine** that stores all of your global settings, and **personas** that represent the individual users who browse the web. Personas are always generated by an engine.

```
engine = PersonaEngine()
persona = engine.persona()
```

By default, personas are single-use and their browsing history will be discarded after your script is run. If you give the persona a name, though, it will save the browsing and recommendation history so you can resume the session later.

```
persona = engine.persona('Mulberry')
```

For example, naming a persona might useful in conjunction with signing in to YouTube, allowing you to imitate a real user watching videos over multiple weeks.

1.2.3 Visiting pages and running commands

If you'd like to visit a single page or run a single action, use `run`:

```
persona.run('https://www.youtube.com/watch?v=hZw23sWlyG0')
```

If you'd like to visit several pages but are too lazy to write a loop, you can use `run_batch`:

```
# Run several actions in a row
persona.run_batch([
    'https://www.youtube.com/watch?v=hZw23sWlyG0',
    'youtube:up_next#10',
    'https://www.youtube.com/watch?v=hZw23sWlyG0'
])
```

Each time you use `.run` it executes an action. An action can be visiting a URL or doing something on the page. For example, liking a video or clicking the next-up video link. Learn more about actions under the bridges section.

1.2.4 Starting and stopping the browser

You can use `with` to automatically start/stop the browser. It makes life easy:

```
# Automatically start/stop the browser
with engine.persona() as persona:
    persona.run('https://www.youtube.com/watch?v=hZw23sWlyG0')
    persona.run('youtube:next_up#3')
```

If you don't use `with`, the browser will automatically launch with your first `.run` statement. You should manually call `.quit()` when you're done:

```
# Visit a page
persona.run('https://www.youtube.com/watch?v=hZw23sWlyG0')

# Quit Chrome
persona.quit()
```

1.2.5 Using the results

History is all of your commands you've run and the pages you've visited, while **recommendations** are what you've been recommended (videos, products, etc, depending on the site you're visiting).

> Right now recommendations also include ads and unrelated promoted content. I'm on the fence about whether they should stay or go.

For convenience, you can use `.to_df()` to see history and recommendations as pandas DataFrames.

```
persona.recommendations.to_df()
persona.history.to_df()
```

If you'd prefer to do your analysis elsewhere, you can save them as CSV files.

```
persona.recommendations.to_csv('recs.csv')
persona.history.to_csv('hist.csv')
```

1.2.6 Headless mode

By default, Chrome runs in **headless** mode, which means you won't see what the browser is doing as it's clicking around and doing things. If we'd like to watch Chrome in action, we can turn off headless mode.

```
engine = PersonaEngine(headless=False)
```

When running in non-headless mode, Persine automatically installs uBlock Origin so we don't have to deal with ads.

Headless mode doesn't support extensions, so by default our invisible Chrome is unfortunately watching ads. We should probably switch to Firefox but it has its own problems.

1.2.7 Bridges

Bridges are site-specific scrapers that tell Persine what to click and what to scrape. They're also in charge of site-specific commands like `youtube:like`.

Currently Persine has a fully-featured bridge for YouTube and an under-development Amazon bridge.

1.3 Bridges and actions

Bridges are site-specific scrapers that tell Persine what to click and what to scrape. In order for Persine to successfully understand a site, it needs a bridge for that site.

When you visit a URL, the bridge for that domain is used to processed the page. Bridges are also in charge of site-specific commands such as `youtube:like`.

Note: If you'd like an action to repeat multiple times you can append `#[NUMBER]` to the action name. For example, `youtube:next_up#50` will watch the next fifty "next up" videos.

1.3.1 YouTube

The *YoutubeBridge* is a general-purpose YouTube scraper. Can pull recommendations from the homepage, search results, and video pages.

Actions

Action	Description
youtube:homepage	Visit youtube.com
youtube:search?QUERY	Search YouTube for the specified term
youtube:next_up	When on a video page, click the "next up" video
youtube:like	Click the like button
youtube:dislike	Click the dislike button
youtube:subscribe	Click the subscribe button
youtube:unsubscribe	Click the unsubscribe button
youtube:sign_in	Begin the signin process – you'll need to complete the process manually, but Persine will resume as soon as it notices you're logged in

1.3.2 Amazon

The *AmazonBridge* is still in development, but here's what it can do so far.

Actions

Action	Description
amazon:homepage	Visit amazon.com
amazon:search?QUERY	Search Amazon for the specified term
amazon:asin?ASIN	Visit the page for a given ASIN

1.3.3 Adding new bridges

Bridges are easy to add! Take a look at the Amazon one as an example – all you really need to implement to build your own is `.run` that returns data from the page. It's easy to scrape using Selenium or by running JavaScript on the page itself and returning the results.

1.4 API Documentation

Persine is built on a triumvirate of pieces:

1. The *PersonaEngine*, which more or less stores the settings for everything you'd like to do, and serves as the entry point for all of your adventures.
2. *Persona*, which are the users that interact with websites. Each persona is attached to a Chrome profile, so browsing history, cookies, etc can all carry over to subsequent sessions. (Note that by default sessions do *not* carry information over)

3. *bridges*, which are the interfaces between Persine and the data on the website. They're the scrapers that pull the recommendations off of the page, and the tools that enables you to write shortcuts like `youtube:search?kittens`

1.4.1 PersonaEngine

class persine.**PersonaEngine**(*height=1200*, *width=1600*, *screenshot_scale=0.5*, *screenshot=None*, *html=None*, *compress_html=True*, *cache_dir=None*, *data_dir=None*, *headless=False*, *driver=None*, *resume=False*, *ublock=False*)

Bases: `object`

PersonaEngine is used to generate personas. You can think of it as a place to store all of your settings.

Parameters

- **height** (*int*) – Height of the browser window
- **weight** (*int*) – Width of the browser window
- **screenshot_scale** (*float*) – Scaling factor for saved screenshots
- **screenshot** (*Union[str, list]*) – Whether screenshots are saved, and whether they go to history or to disk
- **html** (*Union[str, list]*) – Whether HTML is saved, and whether it goes to history or to disk
- **compress_html** (*boolean*) – Whether HTML should be compressed or not before saving to the history
- **cache_dir** (*str*) – Where to save on-disk screenshots and HTML files
- **data_dir** (*str*) – Root directory where persona data (Chrome profiles) are stored
- **headless** (*boolean*) – Whether to start the browser in headless mode
- **driver** – WebDriver to use instead of starting a new one
- **resume** (*boolean*) – Whether to pick up where the previous run left off.
- **ublock** (*boolean*) – Whether to automatically install uBlock Origin

get_driver_options(*user_data_dir=None*)

Create the options necessary to start the appropriate webdriver.Chrome instance

Returns webdriver.ChromeOptions

get_state(*driver*, *url*)

Get the current state of the page.

Returns A representation of the current page (key, action, url, etc)

Return type dict

launch(*user_data_dir=None*)

Launches a Chrome instance.

Returns webdriver.Chrome

persona(*name=None*, *resume=False*)

Initializes a persona with the given name.

Returns The persona initialized by the engine.

Return type *Persona*

run(*driver*, *url*)
Runs a command through the appropriate bridge.

Returns A single state representation. Will return a list of state representations if it's a multi-step command. For example, youtube:next_up#30 to hit 'next up' 30 times

Return type Union[dict, list(dict)]

take_screenshot(*driver*)
Take a screenshot of the current window.

Returns The resized screenshot

Return type Image

1.4.2 Persona

class persine.**Persona**(*engine*, *name=None*, *history_path=None*, *user_data_dir=None*, *resume=False*, *overwrite=False*)
Bases: object

The Persona represents a single user. If it is given a name, it is associated with an individual Chrome profile.

Parameters

- **engine** (PersonaEngine) – The engine to associate with this persona
- **name** (*str*) – The name to be given to this profile. If not named, an empty profile is used.
- **history_path** (*str*) – Path to the JSON file that holds this persona's action/browsing history
- **user_data_dir** (*str*) – If specified, load the Chrome profile from this folder
- **resume** (*boolean*) – Whether this persona should resume a previous persona with the same name. If False, the previous Chrome profile is deleted.
- **overwrite** (*boolean*) – Whether to prompt the user when overwriting a previous persona's Chrome profile (see resume)

clear()
Deletes all previous data for that Chrome profile, including history file and user_data_dir

launch()
Launches a browser through PersonaEngine

load_history()
Loads the browsing/command history from a file

quit()
Quits the browser

run(*url*, *notes=None*)
Runs a single command and updates the history :param url: The action to run or URL to visit :type url: str :param notes: Additional information to include in the history row :type notes: dict

Returns

A single state representation. Will return a list of state representations if it's a multi-step command. For example, youtube:next_up#30 to hit 'next up' 30 times

Return type Union[dict, list(dict)]

run_batch(*urls*)
Run a series of commands

save_history()
Saves the browsing/command history to a file

update_history(*state*, *notes=None*)
Updates history/recommendations lists with the given state

1.4.3 Bridges

class persine.bridges.**AmazonBridge**(*driver*)
Bases: persine.bridges.bridge.BaseBridge

A bridge that interacts with and scrapes Amazon

get_data()
Return import data from the page, as well as a list of the recommendations

Returns Representation of the page

Return type dict

run(*url*)
Run an action/visit a URL.

Returns Representation of the page

Return type dict

class persine.bridges.**BaseBridge**(*driver*)
Bases: object

A completely useless Bridge that at least shows you what they're supposed to implement

Parameters **driver** – A Selenium WebDriver used to navigate

get_data()
Return import data from the page, as well as a list of the recommendations

Returns Representation of the page

Return type dict

run(*url*)
Run an action/visit a URL.

Returns Representation of the page

Return type dict

class persine.bridges.**YoutubeBridge**(*driver*)
Bases: persine.bridges.bridge.BaseBridge

A bridge that interacts with and scrapes YouTube

get_data()
Return import data from the page, as well as a list of the recommendations

Returns Representation of the page

Return type dict

run(*url*)
Run an action/visit a URL.

Returns Representation of the page

Return type dict

- modindex

PYTHON MODULE INDEX

p

INDEX