partridge Documentation

Release 1.1.1

Danny Whalen

Nov 17, 2020

Contents

1 Partridge **3**
- 1.1 Philosphy . . . 3
- 1.2 Installation . . . 4
- 1.3 Usage . . . 4
- 1.4 Features . . . 5
- 1.5 Thank You . . . 6

2 Installation **7**
- 2.1 Stable release . . . 7
- 2.2 From sources . . . 7

3 Usage **9**

4 Contributing **11**
- 4.1 Types of Contributions . . . 11
- 4.2 Get Started! . . . 12
- 4.3 Pull Request Guidelines . . . 13
- 4.4 Tips . . . 13

5 History **15**
- 5.1 1.1.1 (2019-09-13) . . . 15
- 5.2 1.1.0 (2019-02-21) . . . 15
- 5.3 1.0.0 (2018-12-18) . . . 15
- 5.4 0.11.0 (2018-08-01) . . . 16
- 5.5 0.10.0 (2018-04-30) . . . 16
- 5.6 0.9.0 (2018-03-24) . . . 16
- 5.7 0.8.0 (2018-03-14) . . . 16
- 5.8 0.7.0 (2018-03-09) . . . 16
- 5.9 0.6.1 (2018-02-24) . . . 16
- 5.10 0.6.0 (2018-02-21) . . . 17
- 5.11 0.6.0.dev1 (2018-01-23) . . . 17
- 5.12 0.5.0 (2017-12-22) . . . 17
- 5.13 0.4.0 (2017-12-10) . . . 17
- 5.14 0.3.0 (2017-10-12) . . . 17
- 5.15 0.2.0 (2017-09-30) . . . 17
- 5.16 0.1.0 (2017-09-23) . . . 17

6 Indices and tables 19

Contents:

CHAPTER 1

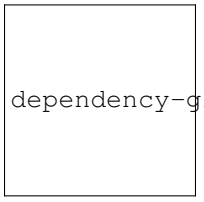
Partridge

Partridge is a Python 3.6+ library for working with GTFS feeds using pandas DataFrames.

Partridge is heavily influenced by our experience at Remix analyzing and debugging every GTFS feed we could find.

At the core of Partridge is a dependency graph rooted at `trips.txt`. Disconnected data is pruned away according to this graph when reading the contents of a feed.

Feeds can also be filtered to create a view specific to your needs. It's most common to filter a feed down to specific dates (`service_id`) or routes (`route_id`), but any field can be filtered.



1.1 Philosphy

The design of Partridge is guided by the following principles:

As much as possible

- Favor speed
- Allow for extension
- Succeed lazily on expensive paths
- Fail eagerly on inexpensive paths

As little as possible

- Do anything other than efficiently read GTFS files into DataFrames
- Take an opinion on the GTFS spec

1.2 Installation

```
pip install partridge
```

GeoPandas support

```
pip install partridge[full]
```

1.3 Usage

Setup

```
import partridge as ptg

inpath = 'path/to/caltrain-2017-07-24/'
```

1.3.1 Inspecting the calendar

The date with the most trips

```
date, service_ids = ptg.read_busiest_date(inpath)
#  datetime.date(2017, 7, 17), frozenset({'CT-17JUL-Combo-Weekday-01'})
```

The week with the most trips

```
service_ids_by_date = ptg.read_busiest_week(inpath)
#  {datetime.date(2017, 7, 17): frozenset({'CT-17JUL-Combo-Weekday-01'}),
#   datetime.date(2017, 7, 18): frozenset({'CT-17JUL-Combo-Weekday-01'}),
#   datetime.date(2017, 7, 19): frozenset({'CT-17JUL-Combo-Weekday-01'}),
#   datetime.date(2017, 7, 20): frozenset({'CT-17JUL-Combo-Weekday-01'}),
#   datetime.date(2017, 7, 21): frozenset({'CT-17JUL-Combo-Weekday-01'}),
#   datetime.date(2017, 7, 22): frozenset({'CT-17JUL-Caltrain-Saturday-03'}),
#   datetime.date(2017, 7, 23): frozenset({'CT-17JUL-Caltrain-Sunday-01'})}
```

Dates with active service

```
service_ids_by_date = ptg.read_service_ids_by_date(path)

date, service_ids = min(service_ids_by_date.items())
#  datetime.date(2017, 7, 15), frozenset({'CT-17JUL-Caltrain-Saturday-03'})

date, service_ids = max(service_ids_by_date.items())
#  datetime.date(2019, 7, 20), frozenset({'CT-17JUL-Caltrain-Saturday-03'})
```

Dates with identical service

```
dates_by_service_ids = ptg.read_dates_by_service_ids(inpath)

busiest_date, busiest_service = ptg.read_busiest_date(inpath)
dates = dates_by_service_ids[busiest_service]

min(dates), max(dates)
#  datetime.date(2017, 7, 17), datetime.date(2019, 7, 19)
```

1.3.2 Reading a feed

```
_date, service_ids = ptg.read_busiest_date(inpath)

view = {
    'trips.txt': {'service_id': service_ids},
    'stops.txt': {'stop_name': 'Gilroy Caltrain'},
}

feed = ptg.load_feed(path, view)
```

Read shapes and stops as GeoDataFrames

```
service_ids = ptg.read_busiest_date(inpath)[1]
view = {'trips.txt': {'service_id': service_ids}}

feed = ptg.load_geo_feed(path, view)

feed.shapes.head()
#       shape_id                                           geometry
#  0  cal_gil_sf  LINESTRING (-121.5661454200744 37.003512297983...
#  1  cal_sf_gil  LINESTRING (-122.3944115638733 37.776439059278...
#  2   cal_sf_sj  LINESTRING (-122.3944115638733 37.776439059278...
#  3  cal_sf_tam  LINESTRING (-122.3944115638733 37.776439059278...
#  4   cal_sj_sf  LINESTRING (-121.9031703472137 37.330157067882...

minlon, minlat, maxlon, maxlat = feed.stops.total_bounds
#  -122.412076, 37.003485, -121.566088, 37.77639
```

1.3.3 Extracting a new feed

```
outpath = 'gtfs-slim.zip'

view = {'trips.txt': {'service_id': ptg.read_busiest_date(inpath)[1]}}

ptg.extract_feed(inpath, outpath, view)
feed = ptg.load_feed(outpath)

assert service_ids == set(feed.trips.service_id)
```

1.4 Features

- Surprisingly fast :)

- Load only what you need into memory
- Built-in support for resolving service dates
- Easily extended to support fields and files outside the official spec (TODO: document this)
- Handle nested folders and bad data in zips
- Predictable type conversions

1.5 Thank You

I hope you find this library useful. If you have suggestions for improving Partridge, please open an issue on GitHub.

CHAPTER 2

Installation

2.1 Stable release

To install partridge, run this command in your terminal:

```
$ pip install partridge
```

This is the preferred method to install partridge, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for partridge can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/remix/partridge
```

Or download the tarball:

```
$ curl  -OL https://github.com/remix/partridge/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use partridge in a project:

```
import partridge
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/remix/partridge/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

partridge could always use more documentation, whether as part of the official partridge docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/remix/partridge/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *partridge* for local development.

1. Fork the *partridge* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/partridge.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv partridge
$ cd partridge/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests:

```
$ flake8 partridge tests
$ python setup.py test or py.test
$ tox
```

   To get flake8, just pip install it into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6+. Check https://travis-ci.org/remix/partridge/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_feed
```

CHAPTER 5

History

5.1 1.1.1 (2019-09-13)

- Improve file encoding sniffer, which was misidentifying some Finnish/emoji unicode. Thanks to @dyakovlev!

5.2 1.1.0 (2019-02-21)

- Add `partridge.load_geo_feed` for reading stops and shapes into GeoPandas GeoDataFrames.

5.3 1.0.0 (2018-12-18)

This release is a combination of major internal refactorings and some minor interface changes. Overall, you should expect your upgrade from pre-1.0 versions to be relatively painless. A big thank you to @genhernandez and @csb19815 for their valuable design feedback. If you still need Python 2 support, please continue using version 0.11.0.

Here is a list of interface changes:

- The class `partridge.gtfs.feed` has been renamed to `partridge.gtfs.Feed`.
- The public interface for instantiating feeds is `partridge.load_feed`. This function replaces the previously undocumented function `partridge.get_filtered_feed`.
- A new function has been added for identifying the busiest week in a feed: `partridge.read_busiest_date`
- The public function `partridge.get_representative_feed` has been removed in favor of using `partridge.read_busiest_date` directly.
- The public function `partridge.writers.extract_feed` is now available via the top level module: `partridge.extract_feed`.

Miscellaneous minor changes:

- Character encoding detection is now done by the `cchardet` package instead of `chardet`. `cchardet` is faster, but may not always return the same result as `chardet`.
- Zip files are unpacked into a temporary directory instead of reading directly from the zip. These temporary directories are cleaned up when the feed is garbage collected or when the process exits.
- The code base is now annotated with type hints and the build runs `mypy` to verify the types.
- DataFrames are cached in a dictionary instead of the `functools.lru_cache` decorator.
- The `partridge.extract_feed` function now writes files concurrently to improve performance.

5.4 0.11.0 (2018-08-01)

- Fix major performance issue related to encoding detection. Thank you to @cjer for reporting the issue and advising on a solution.

5.5 0.10.0 (2018-04-30)

- Improved handling of non-standard compliant file encodings
- Only require functools32 for Python < 3
- `ptg.parsers.parse_date` no longer accepts dates, only strings

5.6 0.9.0 (2018-03-24)

- Improves read time for large feeds by adding LRU caching to `ptg.parsers.parse_time`.

5.7 0.8.0 (2018-03-14)

- Gracefully handle completely empty files. This change unifies the behavior of reading from a CSV with a header only (no data rows) and a completely empty (zero bytes) file in the zip.

5.8 0.7.0 (2018-03-09)

- Fix handling of nested folders and zip containing nested folders.
- Add `ptg.get_filtered_feed` for multi-file filtering.

5.9 0.6.1 (2018-02-24)

- Fix bug in `ptg.read_service_ids_by_date`. Reported by @cjer in #27.

5.10 0.6.0 (2018-02-21)

- Published package no longer includes unnecessary fixtures to reduce the size.
- Naively write a feed object to a zip file with `ptg.write_feed_dangerously`.
- Read the earliest, busiest date and its `service_id`'s from a feed with `ptg.read_busiest_date`.
- Bug fix: Handle `calendar.txt`/`calendar_dates.txt` entries w/o applicable trips.

5.11 0.6.0.dev1 (2018-01-23)

- Add support for reading files from a folder. Thanks again @danielsclint!

5.12 0.5.0 (2017-12-22)

- Easily build a representative view of a zip with `ptg.get_representative_feed`. Inspired by peartree.
- Extract out GTFS zips by agency_id/route_id with `ptg.extract_{agencies,routes}`.
- Read arbitrary files from a zip with `feed.get('myfile.txt')`.
- Remove `service_ids_by_date`, `dates_by_service_ids`, and `trip_counts_by_date` from the feed class. Instead use `ptg.{read_service_ids_by_date,read_dates_by_service_ids, read_trip_counts_by_date}`.

5.13 0.4.0 (2017-12-10)

- Add support for Python 2.7. Thanks @danielsclint!

5.14 0.3.0 (2017-10-12)

- Fix service date resolution for raw_feed. Previously raw_feed considered all days of the week from calendar.txt to be active regardless of 0/1 value.

5.15 0.2.0 (2017-09-30)

- Add missing edge from fare_rules.txt to routes.txt in default dependency graph.

5.16 0.1.0 (2017-09-23)

- First release on PyPI.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search