tidyextractors Documentation

Release 0.2.1

Joel Becker, Jillian Anderson, Steve McColl, John McLevey

Sep 25, 2017

About tidyextractors

CHAPTER 1

Overview: Data Extraction Made Simple

`tidyextractors` makes extracting data from supported sources as painless as possible, delivering a populated Pandas DataFrame in three lines of code. `tidyextractors` was inspired by Hadley Wickham's (2014) paper which introduces "tidy data" as a conceptual framework for data preparation.

Features

- Extracts data with minimal effort.
- Creates readable code that requires minimal explanation.
- Exports Pandas Dataframes to maximize compatibility with the Python data science ecosystem.

Data Sources Implemented

`tidyextractors` currently has submodules for extracting data from the following sources:

- Local Git Repositories
- Twitter User Data (including Tweets) using the Twitter API
- Emails stored in the `mbox` file format.

CHAPTER 2

What is Tidy Data?

Hadley Wickham (2014) introduced "tidy data" to describe data that has been cleaned and reshaped in a way that is ready for analysis. The concept of tidy data inspired `tidyextractors`, which provides a convenient interface for extracting data in a tidy format. However, what is a tidy format?

Our Definition of Tidy Data

We consider data to be "tidy" if it satisfies the following constraints:

- Data values are atomic. No cell contains a collection of items (e.g. a list, set, or dictionary).
- Each row is a single observation. This is to say that each row represents a single "entity" (e.g. such as a commit, a change to a file, or a tweet) which can be uniquely identified by a primary key (e.g. MessageID for an email, or MessageID and recipient for an email "send").
- Each column is a single variable.

This definition intentionally allows for a certain degree of data redundancy, which would be eliminated in traditional database normalizations, such as BCNF.

Choosing Output Formats

We have a few guiding principals for deciding which output formats to implement for a given extractor:

- If an entity would have its own table in a normalized database, it should be available as an output format.
- Dataset should include all variables that have meaningful information about the table's defining unit of observation, even if this data may be redundant between rows.
- In general, more data is preferred to less data, so long as it is meaningful. It is easier to drop data than to integrate data.

Not Your Grandfather's Tidy Data

If you are familiar with Hadley's paper, you may notice that our definition is different from his. We did this because we find the original definition to be self-contradictory. Hadley claims that tidy data is "Codd's 3rd normal form, but with the constraints framed in statistical language, and the focus put on a single dataset rather than the many connected datasets common in relational databases." However, Codd's 3rd normal form *requires* multiple tables.

We agree with Hadley's claim that single table datasets are optimal for data analysis. We also agree with his practice of preferring single datasets at the cost of some data redundancy. We feel that our definition of tidy data is in the spirit of Hadley's original paper.

CHAPTER 3

Installation

With `pip`

Run `pip install tidyextractors` in your terminal. This will install the most recently released stable version of `tidyextractors`.

The Hard Way

Clone the project's GitHub repository and run `pip install .` in the cloned directory.

Note this will install the current state of the project as it is on GitHub. This may not correspond to an official release and is not guaranteed to be stable.

CHAPTER 4

Git Repository Data Extraction

The `tidyextractors.tidygit` submodule lets you extract Git log data from a local Git repository. This page will guide you through the process.

A Minimal Code Example

```
from tidyextractors.tidygit import GitExtractor

# Extract data from a local Git repo
gx = GitExtractor('./your/repo/dir/')

# Commit data in a Pandas DataFrame.
commits_df = gx.commits(drop_collections=True)

# Commit/file keyed change data in a Pandas DataFrame
changes_df = gx.changes()
```

Step 1: Prepare Your Git Repo

All you need to get started is the path to a local Git repository. If you want to extract data from a repository hosted on GitHub, download or clone the repository to your computer.

Step 2: Extract Data

You can extract data from any local Git repository using the `GitExtractor`:

```
from tidyextractors.tidygit import GitExtractor

gx = GitExtractor('./your/repo/dir/')
```

You may need to wait while the data is being extracted, but all the data is now stored inside the extractor object. You just need a bit more code to get it in your preferred format.

Step 3: Get Pandas Data

Now, you can call a `GitExtractor` method to return data in a Pandas DataFrame.

```
# Commit data in a Pandas DataFrame.
commits_df = gx.commits(drop_collections=True)

# Commit/file keyed change data in a Pandas DataFrame
changes_df = gx.changes()
```

Note: `GitExtractor.commits()` drops columns with collections of data in cells (i.e. `list`, `set`, and `dicts`) because "tidy data" requires only atomic values in cells. If you don't want data dropped, change the optional `drop_collections` argument to false.

CHAPTER 5

Mbox Data Extraction

Mbox is a file format used to store mailbox data on Unix operating systems. The `tidyextractors.tidymbox` submodule lets you extract user data from Mbox files with minimal effort. This page will guide you through the process.

A Minimal Code Example

```
from tidyextractors.tidymbox as MboxExtractor

# Extracts all mbox files in this directory.
mx = MboxExtractor('./your/mbox/dir/')

# Email messages in a Pandas DataFrame.
email_df = mx.emails(drop_collections=True)

# MessageID/receiver keyed Pandas DataFrame.
sends_df = mx.sends()
```

Step 1: Prepare Your Mbox Files

You can extract data from a single Mbox file, or multiple Mbox files. However, all these files must be in a single directory:

```
ls -1 ./your/mbox/dir/
file1.mbox
file2.mbox
file3.mbox
```

Step 2: Extract Data

Once you have consolidated your Mbox files, you can extract data from them using the `MboxExtractor`:

```
from tidyextractors.tidymbox as MboxExtractor

# All mbox files in the directory
mx = MboxExtractor('./your/mbox/dir/')

# Only one mbox file
mx = MboxExtractor('./your/mbox/dir/file1.mbox')
```

You may need to wait while the data is being extracted, but all the data is now stored inside the extractor object. You just need a bit more code to get it in your preferred format.

Step 3: Get Pandas Data

Now, you can call an `MboxExtractor` method to return data in a Pandas DataFrame.

```
# Email messages in a Pandas DataFrame.
email_df = mx.emails(drop_collections=True)

# MessageID/receiver keyed Pandas DataFrame.
sends_df = mx.sends()
```

Note: `MboxExtractor.emails()` drops columns with collections of data in cells (i.e. `list`, `set`, and `dicts`) because "tidy data" requires only atomic values in cells. If you don't want data dropped, change the optional `drop_collections` argument to false.

Note: This submodule's internals were adapted from Phil Deutsch's mbox-to-pandas script with his permission.

CHAPTER 6

Twitter Data Extraction

The `tidyextractors.tidytwitter` submodule lets you extract user data from Twitter with minimal effort. This page will guide you through the process.

A Minimal Code Example

```
from tidyextractors.tidytwitter import TwitterExtractor

# Your Twitter API credentails. See below for how to get them!
credentials = {
  'access_token': '',
  'access_secret': '',
  'consumer_key': '',
  'consumer_secret': ''
}

# A list of users for data extraction.
users = ['user1','user2','user3']

# Extract Twitter data.
tx = TwitterExtractor(users, extract_tweets=True, **credentials)

# Twitter user profile data in a Pandas DataFrame
user_df = tx.users(drop_collections=True)

# User/tweet keyed Pandas DataFrame
tweet_df = tx.tweets()
```

Step 1: Get API Credentials

To extract data using the Twitter API, you will first need to obtain API credentials. Your API credentials contain four pieces of information:

- `access_token`
- `access_secret`
- `consumer_key`
- `consumer_secret`

To get these credentials, check out the Twitter developer documentation: https://dev.twitter.com/oauth/overview/application-owner-access-tokens

Step 2: Extract Data

Once you have your API credentials, you can extract user data with the `TwitterExtractor`:

> **Warning:** The Twitter API enforces rate limits, so be careful when downloading large amounts of data. For a raw report on your remaining limit, call `tx._api.rate_limit_status()` after extraction.

Note: As per the limit imposed by the Twitter API, only the 3,200 most recent tweets will be downloaded for each user.

```
from tidyextractors.tidytwitter import TwitterExtractor

credentials = {
  # Randomly generated example credentials for demonstration only
  'access_token': '985689236-R0EjHQJZLya6gb82R5g8Odb4UMwkhQy4Q2AxzBnB',
  'access_secret': 'CVuVV0LSf74PQt2HH6zt08aeumGdMvlZtKF7BbHvRmX4r',
  'consumer_key': 'F47AzSRag0KvVFG4eJYexuDqB',
  'consumer_secret': 'lovnyqIA1oKs0jI4A27VXLLSUWrKc0hnNzyTu39NWIjSiq1xxj'
}


# User names may have leading "@" but this is not required.
users = ['user1','user2','user3']

# Users' tweets are extracted by default, but this may be disabled.
tx = TwitterExtractor(users, extract_tweets=True, **credentials)
```

You may need to wait while the data is being extracted, but all the data is now stored inside the extractor object. You just need a bit more code to get it in your preferred format.

Step 3: Get Pandas Data

Now, you can call a `TwitterExtractor` method to return data in a Pandas DataFrame.

```
# Twitter user profile data in a Pandas DataFrame
user_df = tx.users(drop_collections=True)

# User/tweet keyed Pandas DataFrame
tweet_df = tx.tweets()
```

Note: `TwitterExtractor.users()` drops columns with collections of data in cells (i.e. `list`, `set`, and `dicts`) because "tidy data" requires only atomic values in cells. If you don't want data dropped, change the optional `drop_collections` argument to false.

CHAPTER 7

API Documentation

An in-depth look at the `tidyextractors` interface.

BaseExtractor

The `BaseExtractor` is the foundation for all data extractors in this module.

> **Warning:** The `BaseExtractor` should never be created on its own. This page is intended for people who want to contribute to the `tidyextractors` project or just want to understand some of what is going on under the hood.

For developers, note that the internal interface is not documented on this site. See the source code for the full documentation of the internal interface.

class `tidyextractors.`**`BaseExtractor`**(*source*, *auto_extract=True*, **args*, ***kwargs*)

BaseExtractor defines a basic interface, initialization routine, and data manipulation tools for extractor subclasses.

`expand_on`(*col1*, *col2*, *rename1=None*, *rename2=None*, *drop=[]*, *drop_collections=False*)

Returns a reshaped version of extractor's data, where unique combinations of values from col1 and col2 are given individual rows.

Example function call from `tidymbox`:

```
self.expand_on('From', 'To', ['MessageID', 'Recipient'], rename1='From',␣
↪rename2='Recipient')
```

Columns to be expanded upon should be either atomic values or dictionaries of dictionaries. For example:

Input Data:

col1 (Atomic)	col2 (Dict of Dict)
value1	{valueA : {attr1: X1, attr2: Y1}, valueB: {attr1: X2, attr2: Y2}
value2	{valueC : {attr1: X3, attr2: Y3}, valueD: {attr1: X4, attr2: Y4}

Output Data:

col1_extended	col2_extended	attr1	attr2
value1	valueA	X1	Y1
value1	valueB	X2	Y2
value2	valueA	X3	Y3
value2	valueB	X4	Y4

Parameters

- **col1** (*str*) – The first column to expand on. May be an atomic value, or a dict of dict.
- **col2** (*str*) – The second column to expand on. May be an atomic value, or a dict of dict.
- **rename1** (*str*) – The name for col1 after expansion. Defaults to col1_extended.
- **rename2** (*str*) – The name for col2 after expansion. Defaults to col2_extended.
- **drop** (*list*) – Column names to be dropped from output.
- **drop_collections** (*bool*) – Should columns with compound values be dropped?

Returns pandas.DataFrame

raw(*drop_collections=False*)

Produces the extractor object's data as it is stored internally.

Parameters **drop_collections** (*bool*) – Defaults to False. Indicates whether columns with lists/dicts/sets will be dropped.

Returns pandas.DataFrame

GitExtractor API Documentation

class tidyextractors.tidygit.**GitExtractor**(*source*, *auto_extract=True*, **args*, ***kwargs*)

The GitExtractor class is for extracting data from local git repositories. This class has methods for outputting data into the changes and commits tidy formats, and a raw untidy format.

Parameters

- **source** (*str*) – The path to a local git repository
- **auto_extract** (*bool*) – Defaults to True. If True, data is extracted automatically. Otherwise, extraction must be initiated through the internal interface.

changes()

Returns a table of git log data, with "changes" as rows/observations.

Note: drop_collections is not available for this method, since there are no meaningful collections to keep.

Returns pandas.DataFrame

commits(*drop_collections=True*)
Returns a table of git log data, with "commits" as rows/observations.

Parameters **drop_collections** (*bool*) – Defaults to True. Indicates whether columns with lists/dicts/sets will be dropped.

Returns pandas.DataFrame

raw(*drop_collections=False*)
Produces the extractor object's data as it is stored internally.

Parameters **drop_collections** (*bool*) – Defaults to False. Indicates whether columns with lists/dicts/sets will be dropped.

Returns pandas.DataFrame

`MboxExtractor` API Documentation

class tidyextractors.tidymbox.**MboxExtractor**(*source*, *auto_extract=True*, **args*, ***kwargs*)
The `MboxExtractor` class is for extracting data from local Mbox files. This class has methods for outputting data into the `emails` and `sends` tidy formats, and a raw untidy format.

Parameters

- **source** (*str*) – The path to either a single mbox file or a directory containing multiple mbox files.
- **auto_extract** (*bool*) – Defaults to True. If True, data is extracted automatically. Otherwise, extraction must be initiated through the internal interface.

emails(*drop_collections=True*)
Returns a table of mbox message data, with "messages" as rows/observations.

Parameters **drop_collections** (*bool*) – Defaults to True. Indicates whether columns with lists/dicts/sets will be dropped.

Returns pandas.DataFrame

raw(*drop_collections=False*)
Produces the extractor object's data as it is stored internally.

Parameters **drop_collections** (*bool*) – Defaults to False. Indicates whether columns with lists/dicts/sets will be dropped.

Returns pandas.DataFrame

sends()
Returns a table of mbox message data, with "sender/recipient" pairs as rows/observations.

Note: Rows may have a recipient from either "TO" or "CC". SendType column specifies this for each row.

Note: drop_collections is not available for this method, since there are no meaningful collections to keep.

Returns pandas.DataFrame

Note: This submodule's internals were adapted from Phil Deutsch's mbox-to-pandas script with his permission.

`TwitterExtractor` API Documentation

class `tidyextractors.tidytwitter.`**`TwitterExtractor`**(*source*, *auto_extract=True*, **args*, ***kwargs*)

The `TwitterExtractor` class is for extracting user data from Twitter. This class has methods for outputting data into the `users` and `tweets` tidy formats, and a raw untidy format.

Parameters

- **`source`** (*`list`*) – A list of user screen name strings.
- **`auto_extract`** (*`bool`*) – Defaults to True. If True, data is extracted automatically. Otherwise, extraction must be initiated through the internal interface.
- **`access_token`** (*`str`*) – One of four required keyword arguments that make up a complete set of Twitter API credentials.
- **`access_secret`** (*`str`*) – One of four required keyword arguments that make up a complete set of Twitter API credentials.
- **`consumer_key`** (*`str`*) – One of four required keyword arguments that make up a complete set of Twitter API credentials.
- **`consumer_secret`** (*`str`*) – One of four required keyword arguments that make up a complete set of Twitter API credentials.

`raw`(*drop_collections=False*)

Produces the extractor object's data as it is stored internally.

Parameters **`drop_collections`** (*`bool`*) – Defaults to False. Indicates whether columns with lists/dicts/sets will be dropped.

Returns pandas.DataFrame

`tweets`()

Returns a table of Twitter user data, with "tweets" as rows/observations.

Note: drop_collections is not available for this method, since there are no meaningful collections to keep.

Returns pandas.DataFrame

`users`(*drop_collections=True*)

Returns a table of Twitter user data, with "users" as rows/observations.

Parameters **`drop_collections`** (*`bool`*) – Defaults to True. Indicates whether columns with lists/dicts/sets will be dropped.

Returns pandas.DataFrame

CHAPTER 8

Contributing

Our hope is that `tidyextractors` will expand to include numerous new data sources. If there is a particular kind of data you are interested in extracting, or which to contribute to the package, please contact Joel Becker (mail@joelbecker.ca) or Jillian Anderson (jillianderson8@gmail.com) and we will respond ASAP.

Creating an Extractor

Contributing a new extractor is relatively simple. Broadly speaking, you need to create a submodule with an extractor class inheriting from `BaseExtractor`. To create this class (e.g. `NewExtractor`) you need to do the following:

- Define a `NewExtractor._extract` method, which should extract data and assign it to `NewExtractor._data`. This method will be called by `BaseExtractor.__init__` during initialization.
- Create a method to return each data format (e.g. `commits`, `changes`).

Python Module Index

t

Index