tabulog

Release 0.1.1

Feb 21, 2020

Contents

Parsing Semi-Structured Log Files into Tabular Format

CHAPTER 1

Introduction to Tabulog

Tabulog is a flexible, powerful framework for parsing log files, specifically designed for web logs (such as the access.log files created by Apache), with the final output being in a tabular format.

Parsing logs with Tabulog requires two things: a template, and a list of "parser classes."

1.1 Tabulog Templates

Inspired by Python's Jinja2 templates, Tabulog templates use a human-readable format mixing literal text with code. Code is being used extremely loosely here, as you will see that the 'code' in our templates is not actually R code.

The easiest place to start is with an example. Let's say you have a simple log file that looks like this:

```
10.0.0.8 - - [2019-01-01:10:58:12 -500] "https://mysite.com/index.html"
173.28.102.33 - - [2019-01-01:10:58:25 -500] "https://mysite.com/login"
...
```

We can see the log file here holds a certain format, specifically:

```
<ip address> - - [<datetime>] "<url>"
```

The Tabulog template to parse such a file looks like this

```
{{ ip ip_address }} - - [{{ Date date_time }}] "{{ url URL }}"
```

Each set of curly brackets represents an instance of a *class*, and is declared in the C style of `class var_name`. So in the template above, `{{ ip ip_address }}` is really saying "In this spot, look for an ip, and call it `ip_address`."

You may ask, how does the Tabulog know what an ip address *is*? Which is where we are introduced to *parser classes*.

1.2 Parser Classes

In order to know what to look for in each field of our template, Tabulog must know what a given class should look like. For this we give it a *parser class*, which is really just a wrapper object for a regular expression.

In the current example with the ip address, we would tell Tabulog that the ip class is represented by the Perl regular expression: `[0-9]{1,3}(\.[0-9]{1,3}){3}`. When Tabulog parsed the log file, it would look for a match on that expression in that spot, and raise a warning if it didn't find one.

1.2.1 Parser Formatters

Once a field is parsed , you may want to further transform or *format* the text. For example, you may want to cast an integer. This is achieved using formatters.

When a parser object is created, an optional formatter can be passed. This is simply a function that taking the extracted field of text and returining the transformed field.

Tabulog as a framework is designed to be language-agnostic, so the ideas of templates and parser classes here will be portable between languages. Formatters, however, are languagespecific and must be implemented in the language being used.

CHAPTER 2

Installation Instructions

You can install tabulog from PyPI or CRAN as follows

2.1 Python

```
pip install tabulog
```

2.2 R

```
install.packages('tabulog')
```

CHAPTER 3

Examples

Examples are provided for each language

CHAPTER 4

Contributing Guidelines

Contributing guidelines can be found in CONTRIBUTING.md

CHAPTER 5

Notes

5.1 Escape characters

The only characters that need to be escaped in templates are curly braces (even single ones). Usually a backslash should be sufficient `'\{'`, but the html-style escapes `'&#123;'` and `'&#125;'` are also included as valid syntax for any edge cases that may arise.

CHAPTER 6

Python Package

This page contains documentation for the Python package

6.1 Contents

6.1.1 An Example in Python

Let's again say you have the example logs in the file `accesslog.txt`.

```
10.0.0.8 - - [2019-01-01:10:58:12 -0500] "https://mysite.com/index.html"
173.28.102.33 - - [2019-01-01:10:58:25 -0500] "https://mysite.com/login"
```

We first define the template.

```
template = '{{ ip ip_address }} - - [{{ date date_time }}] "{{ url URL }}"'
```

We then need to define our classes. `ip` and `url` are builtins with the package, but dates come in a variety of formats so we must explicitly define ours here. Note you can see all builtins using `default_classes()`

```
import tabulog, datetime

date_parser = tabulog.Parser(
  '[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}[ ][\\-\\+][0-9]{4}',
  lambda x:datetime.datetime.strptime(x, '%Y-%m-%d:%H:%M:%S %z'),
  name = 'date'
)
date_parser
```

```
Parser('[0-9]{4}\-[0-9]{2}\-[0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}[ ][\-\+][0-9]{4}',
↪<function <lambda> at 0x7f7a07574e18>, 'date')
```

```
for key in ['ip', 'url']:
  print(tabulog.default_classes()[key])
```

```
Parser('[0-9]{1,3}(\.[0-9]{1,3}){3}', <function <lambda> at 0x7f7a06c896a8>, 'ip')
Parser('(-|(?:http(s)?:\/\/)?[\w.-]+(?:\.[\w\.-]+)+[\w\-\._~:/?#[\]@!\$&\'\(\)\*\+,;=.
↪]+)', <function <lambda> at 0x7f7a06c896a8>, 'url')
```

Both `ip` and `url` require no formatting, so they have the identity function, (`lambda x:x` in python), as their formatter.

To get our final output in tabular format, we first combine everything into a `Template` object.

```
# We only need to pass our custom date parser class, the defaults will be included.
T = tabulog.Template(
  template_string = template,
  classes = [date_parser]
)
T
```

```
Template("{{ ip ip_address }} - - [{{ date date_time }}] \"{{ url URL }}\"", classes␣
↪= ...)
```

Note that we only had to pass our custom class `date`. The builtin classes `ip` and `url` were included by default.

Finally, we can read in our log file, and call the `tabulate` function in our `Template` object. The final output is a Pandas DataFrame.

```
with open('accesslog.txt', 'r') as f:
  logs = f.read().split('\n')[:-1]

T.tabulate(logs)
```

```
      ip_address                 date_time                              URL
0       10.0.0.8 2019-01-01 10:58:12-05:00  https://mysite.com/index.html
1  173.28.102.33 2019-01-01 10:58:25-05:00       https://mysite.com/login
```

A more elegant and portable way of completing this task would be to define the template and the custom class in the same file, which can be ported to other Tabulog libraries in other languages, leaving only the formatters to be defined in the R script.

First, we define the `template` and the `classes` in a yaml file

```
~$ cat accesslog_template.yml
```

```
template: '{{ ip ip_address }} - - [{{ date date_time }}] "{{ url URL }}"'
classes:
  date: '[0-9]{4}\-[0-9]{2}\-[0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}[ ][\-\+][0-9]{4}'
```

Next, we define the formatters for each of our classes. Here we only have one, but we still put it in a named list, with the name matching the name of the class in the template file.

```
formatters = {
  'date': lambda x:datetime.datetime.strptime(x, '%Y-%m-%d:%H:%M:%S %z')
}
```

Next, we make create our template again, this time using the `file` argument.

```
T = tabulog.Template(
  file = 'accesslog_template.yml',
  formatters = formatters
)
T
```

```
Template("{{ ip ip_address }} - - [{{ date date_time }}] \"{{ url URL }}\"", classes
↪= ...)
```

Again, we get our final output with the same call to `tabulate`.

```
T.tabulate(logs)
```

```
      ip_address                 date_time                              URL
0       10.0.0.8 2019-01-01 10:58:12-05:00  https://mysite.com/index.html
1  173.28.102.33 2019-01-01 10:58:25-05:00       https://mysite.com/login
```

6.1.2 Python Package Docs

Templates

class tabulog.**Template**(*template_string=None*, *file=None*, *classes=[]*, *formatters={}*)

This is the core processor of logs. A Template object stores all the necessary information about the tabulog template string and the parser classes used within that template.

Templates can be created either by passing the tabulog template string and parser classes directly, or by storing the template string and class definitions in a single yaml file.

Attributes:

- template (str): tabulog template string
- classes (dict): :class:Parser objects to be used with the fields in the template

__init__(*template_string=None*, *file=None*, *classes=[]*, *formatters={}*)

Template constructor

Args:

- tempalte_string (str): The tabulog template string
- file (str): Name of a YAML file containing the tabulog string and class definitions
- classes (list): List of Parser objects to be used with template. Ignored if using a template file.
- formatters (dict): Dictionary of formatter functions to associate with the classes defined in *file*. Ignored if using a template_string.

tabulate(*text*)

Tabulate a list of strings (representing lines in a log file) into a Pandas DataFrame.

Args:

- text(list): List of strings to be tabulated.

```
>>> t.tabulate('https://www.example.com/ - 200')
                       url  status
0  https://www.example.com/     200
```

Parsers

class tabulog.**Parser**(*pattern*, *formatter=<function <lambda>>*, *name=None*)

This is a helper class that represents a given field in the tabulog template.

Parser objects have a regex string (pattern) to match for in the record, a formatter function which takes the extracted text for the field and modifies it in some way (ex. cast to integer, make lowercase, etc.), and an optional name.

Attributes:

- pattern (str): regex string to match on in the template
- formatter (callable): Callable (usually a function or lambda expression) to modify the field after extraction
- name (str): optional name for identification

__init__(*pattern*, *formatter=<function <lambda>>*, *name=None*)

Parser constructor

Args:

- pattern (str): regex string to match on in the template
- formatter (callable): Callable (usually a function or lambda expression) to modify the field after extraction
- name (str): optional name for identification

Other Functions

tabulog.**default_classes**()

A dictionary of default Parser classes provided 'out-of-the-box'.

By 'classes' here we mean Parser objects that come predefined with a defining regex string, and possibly a meaningful formatter function.

Returns: A dictionary of Parser objects

```
>>> default_classes()
{'ip': Parser('[0-9]{1,3}(\.[0-9]{1,3}){3}', <function <lambda> at 0x7f492c4426a8>
↪, 'ip')...
```

CHAPTER 7

R Package

This page contains links to all the R documentaion (found on CRAN).

7.1 Contents

7.1.1 An Example in R

Let's again say you have the example logs in the file `accesslog.txt`.

```
log_file <- 'vignettes/accesslog.txt'
cat(readr::read_file(log_file))
```

```
10.0.0.8 - - [2019-01-01:10:58:12 -0500] "https://mysite.com/index.html"
173.28.102.33 - - [2019-01-01:10:58:25 -0500] "https://mysite.com/login"
```

We first define the template as before.

```
template <- '{{ ip ip_address }} - - [{{ date date_time }}] "{{ url URL }}"'
```

We then need to define our classes. `ip` and `url` are builtins with the package, but dates come in a variety of formats so we must explicitly define ours here. Note you can see all builtins using `default_classes()`

```
date_parser <- parser(
  '[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}[ ][\\-\\+][0-9]{4}',
  function(x) lubridate::as_datetime(x, format = '%Y-%m-%d:%H:%M:%S %z'),
  name = 'date'
)
date_parser
```

```
## Parser: date
## ------------
## Matches:
```

(continues on next page)

(continued from previous page)

```
##   [0-9]{4}\-[0-9]{2}\-[0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}[ ][\-\+][0-9]{4}
## Formatter:
##   function (x)
##   lubridate::as_datetime(x, format = "%Y-%m-%d:%H:%M:%S %z")
```

```
default_classes()[c('ip', 'url')]
```

```
## $ip
## Parser: ip
## ----------
## Matches:
##   [0-9]{1,3}(\.[0-9]{1,3}){3}
## Formatter:
##   .Primitive("(")
##
## $url
## Parser: url
## -----------
## Matches:
##   (-|(?:http(s)?:\/\/)?[\w.-]+(?:\.[\w\.-]+)+[\w\-\._~:/?#[\]@!\$&'\(\)\*\+,;=.]+)
## Formatter:
##   .Primitive("(")
```

Both `ip` and `url` require no formatting, so they have the identity function, (`(` in R), as their formatter.

To get our final output in tabular format, we simply make the follow call to `parse_logs`.

```
# Naming the date_parser 'date' in the list tells Tabulog to use it to parse
# the field with class 'date' in the template.
parse_logs(readLines(log_file), template, classes = list(date = date_parser))
```

```
##      ip_address           date_time                          URL
## 1      10.0.0.8 2019-01-01 15:58:12 https://mysite.com/index.html
## 2 173.28.102.33 2019-01-01 15:58:25      https://mysite.com/login
```

Note that we only had to pass our custom class `date`. The builtin classes `ip` and `url` were included by default.

A more elegant and portable way of completing this task would be to define the template and the custom class in the same file, which can be ported to other Tabulog libraries in other languages, leaving only the formatters to be defined in the R script.

First, we define the `template` and the `classes` in a yaml file

```
template_file <- 'vignettes/accesslog_template.yml'
cat(readr::read_file(template_file))
```

```
template: '{{ ip ip_address }} - - [{{ date date_time }}] "{{ url URL }}"'
classes:
  date: '[0-9]{4}\-[0-9]{2}\-[0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}[ ][\-\+][0-9]{4}'
```

Next, we define the formatters for each of our classes. Here we only have one, but we still put it in a named list, with the name matching the name of the class in the template file.

```
formatters <- list(
  date = function(x) lubridate::as_datetime(x, format = '%Y-%m-%d:%H:%M:%S %z')
)
```

Finally, we make one call to `parse_logs_file`.

```
parse_logs_file(log_file, template_file, formatters)
```

```
##      ip_address           date_time                           URL
## 1      10.0.0.8 2019-01-01 15:58:12 https://mysite.com/index.html
## 2 173.28.102.33 2019-01-01 15:58:25      https://mysite.com/login
```

Index

Symbols

D

P

T