SandPaper Documentation

Release 0.0.6

Stephen Bunn

Dec 15, 2017

Contents

CHAPTER 1

Documentation

1.1 Getting Started

This module provides basic table-type data normalization that I have personally needed in many different projects over the past couple years. It allows the normalization of table data readable by the popular pyexcel library (mainly `.xlsx`, `.xls`, and `.csv` files).

It uses the concept of rules which can be chained to form a cleaning processes for these files. **This module has a lot of room for improvement**, but it gets the job done that I needed it to. Hopefully I'll continue to contribute rules, features, and more clean functionality as I need it.

1.1.1 Installation

Currently SandPaper is on PyPi and can easily be installed through pip!

```
pip install sandpaper
```

1.1.2 Usage

Using SandPaper is *fairly* simple and straightforward. First things first, in order to normalize any data you have to create an instance of the `SandPaper` object to group together your normalization rules.

```
from sandpaper import SandPaper

# for an explicitly named sandpaper instance
my_sandpaper = SandPaper('my-sandpaper')

# for an implicitly named sandpaper instance
my_sandpaper = SandPaper()
```

Chaining Rules

Now that you have a `SandPaper` instance, you can start chaining in rules that should be applied in order to normalize the data.

Tip: For a full list of available rules, check out the list of rules here.

Rule can be applied by simply chaining the ordered normalization processes directly off of a `SandPaper` isntance.

```
my_sandpaper.strip()
```

This will apply the `strip()` rule to the `my_sandpaper` instance. The way it is now, the `my_sandpaper` instance will strip all whitespace from all values (since no filters were given).

We can add another rule to `my_sandpaper` by simply calling it.

```
my_sandpaper.translate_text({
   r'FL': 'Florida',
   r'NC': 'North Carolina'
}, column_filter=r'state')
```

This will apply the `translate_text()` rule to the `my_sandpaper` instance.

Since the `strip()` rule has already been applied, stripping of all whitespace will occur before this rule is applied. The `translate_text()` rule will substitute the regular expression matches `FL` and `NC` with the values `Florida` and `North Carolina` respectively only in the column matching the filter `state`.

The current state of the `my_sandpaper` instance could have also been initialized in one go using the chaining feature that rules provide.

```
my_sandpaper = SandPaper('my-sandpaper')\
   .strip()\
   .translate_text({
      r'FL': 'Florida',
      r'NC': 'North Carolina'
   }, column_filter=r'state')
```

Applying SandPaper

In order to run this `SandPaper` instance you need to call the `apply()` method to a file.

```
my_sandpaper.apply('/path/to/input_file.csv', '/path/to/output_file.csv')
```

Important: If applying to `.csv` files, unnecessary quotations are implicitly removed as part of the reading and saving processes. Currently there is no way of disabling this… sorry .

Rule Filters

An important thing to note about rules is that every value has to first pass several optional filters if the rule is to be applied to that value.

`column_filter` [regex] A regular expression filter applied to the column name of the value (*must have a match to pass*)

`value_filter` [regex] A regular expression filter applied to the value (*must have a match to pass*)

`callable_filter` [callable] A callable reference that is executed for each value (*must evaluate to true to pass*)

> **Note:** This callable should expect to receive the parameters `record`, `column` in that order, as well as any specified rule kwargs. The callable should return a boolean value which is True if the rule should be applied, otherwise False.

These filters are processed in the order presented and are completely optional. **If no filters are specified, then the rule is applied.**

Saving SandPapers

It is possible to export a *`SandPaper`* instance using the `__json__()` function. This exports the configuration of the intance to a dictionary which is suitable for json serialization.

```
serialized = my_sandpaper.__json__()
```

This exported format can be used to bootstrap a new *`SandPaper`* instance by providing the serialization to the *`from_json()`* method.

```
new_sandpaper = SandPaper.from_json(serialized)
```

Important: The json serialization does not store any information about callables. A `UserWarning` is raised during serialization if a callable is found.

```
def _filter_handler(record, column, **kwargs):
   return record[column].lower().startswith('north')

paper = SandPaper().translate_text({
   r'FL': 'Florida',
   r'NC': 'North Carolina'
}, callable_filter=_filter_handler)

paper.__json__()
# raises: UserWarning because of _filter_handler
```

1.1.3 Limitations

Several limitations to the effectiveness of the reading and writing of normalized data still exist within this module. These are described in the subsections below…

One Sheet

SandPaper instance's are really only meant to be applied to one sheet at a time. Of course, this only applies to those table storage types that implement sheets (`.xlsx`, `.xls`, etc…).

It is up to the user to create SandPaper instances as necessary for each sheet that requires cleaning. This module is intended to clean and return parsable normalized data, not do Excell's job for it.

Reading as Records

In order to provide all of the lovely filtering (*Rule Filters*) that make specifying advanced normalization rules much easier, SandPaper reads rows of table type data in as records (`collections.OrderedDict`). This allows us to tie row entries to column names easily but unfortunately causes limitations on the format of data that can be properly read in. The main limitation is that **table sheets with duplicate column names cannot be read properly**.

Because pyexcel reads records as `OrderedDict`, the last column with a duplicate name is the only column considered.

For example the following table data…

my_column	my_column
1	2
3	4

will only output the last `my_column` column (with values 2 and 4) in the resulting `sanded` data. This is because the reading of the record first reads the first column and then overwrites it with the second column.

A fix for this issue is possible, however would cause a lot of refactoring and additional testing which (obviously) has not been done.

Translating Dates

The *`translate_date()`* rule is quite nifty, but also has a couple limitations that need to be considered. Because dates are not a base type, the mentioned rule can sometimes incorrectly interpret strings as dates and apply a date translation where it is not needed. For this reason, it is recommended to also specify at least a `column_filter` for all instances of the rule. A `value_filter` would also help, but causes a lot of extra complexity that is most likely not required.

1.2 Available Rules

Below are a list of available rules that can be attached to a *`SandPaper`* instance. All of these rules first must pass several optional filters discussed in *Rule Filters*.

In the following examples of these rules the symbol represents whitespace.

1.2.1 Value Rules

These rules are applied to every value that passes the specified rule filters documented in *Rule Filters*.

`lower()`

A basic rule that lowercases the text in a value.

```
SandPaper().lower()
```

Input	Output
DATa	data

upper()

A basic rule that uppercases the text in a value.

```
SandPaper().upper()
```

Input	Output
daTa	DATA

capitalize()

A basic rule that capitalizes the text in a value.

```
SandPaper().capitalize()
```

Input	Output
daTa	Data

title()

A basic rule that titlecases the text in a value.

```
SandPaper().title()
```

Input	Output
mY dAta	My Data

lstrip()

A basic rule that strips all *left* whitespace from a value.

```
SandPaper().lstrip()
```

Input	Output
data	data

rstrip()

A basic rule that strips all *right* whitespace from a value.

```
SandPaper().rstrip()
```

Input	Output
data	data

strip()

A basic rule that strips *all* whitespace from a value.

```
SandPaper().strip()
```

Input	Output
data	data

translate_text()

A translation rule that translate regex matches to a specified format.

```
SandPaper().translate_text({
    r'group_(?P<group_id>\d+)$': '{group_id}'
})
```

Input	Output
group_47	47
group_123	123
group_0	0

translate_date()

A translation rule that translate greedily evaluated dates to a specified datetime format.

Note: This rule is very greedy and can potentailly evaluate dates incorrectly. It is **highly recommended** that at the very least a `column_filter` is supplied with this rule.

```
SandPaper().translate_date({
    '%Y-%m-%d': '%Y',
    '%Y-%m': '%Y',
    '%Y': '%Y'
})
```

Input	Output
2017-01-32	2017
2017-01	2017
2017	2017

1.2.2 Record Rules

These rules are applied to every record regardless to most rule filters documented in *Rule Filters*.

add_columns()

Adds a column to every record.

The given dictionary should be a key value pairing where the key is a new column name and the paired value is either a callable, string, or other low level data type for the newly added column's value. If the value is a callable it should expect to receive the record as the only parameter and should return the value desired for the newly added column.

```python
import uuid

def gen_uuid(record):
   return uuid.uuid4()

SandPaper().add_columns({
   'uuid': gen_uuid
})
```

Before		
id	name	value
1	hello	world
2	test	table

After			
id	name	value	uuid
1	hello	world	a6a76896-c33d-4654-afdf-12aa80dd6238
2	test	table	b1e171c2-fee9-4270-96e9-4853c3a6e130

remove_columns()

Removes a column from every record.

```python
SandPaper().remove_columns([
   'name'
])
```

Before		
id	name	value
1	hello	world
2	test	table

After	
id	value
1	world
2	table

keep_columns()

Removes all other columns for every record.

```
SandPaper().keep_columns([
   'id',
   'name'
])
```

Before		
id	name	value
1	hello	world
2	test	table

After	
id	name
1	hello
2	test

rename_columns()

Renames a column for every record.

```
SandPaper().rename_columns([
   'old_name': 'new_name'
])
```

Before		
id	old_name	value
1	hello	world
2	test	table

After		
id	new_name	value
1	hello	world
2	test	table

order_columns()

Reorders columns from every record.

```
SandPaper().order_columns([
   'value',
   'name',
   'id'
])
```

Before		
id	name	value
1	hello	world
2	test	table

After		
value	id	name
world	1	hello
table	2	test

1.3 Changelog

All notable changes to SandPaper will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

1.3.1 *unreleased*

- WIP

1.3.2 0.0.6 (*2017-12-15*)

- added functionality for a `row_filter` callable specified during `apply`
- added `keep_columns` record rule
- added built-in Sphinx theme (borrowed from pocoo click)
- fixed `__json__` export warnings with callables

1.3.3 0.0.5 (*2017-11-03*)

- added enforcement for strict date parsing in `translate_date` rule
- added `rename_columns` and `order_columns` record rules
- fixed the naming of `add_columns` and `remove_columns`
- fixed the messy structure of all rules (cleaner and more intuitive use)
- fixed documentation to match new rule structure

- fixed all existing tests to match new rule structure
- removed the `substitute` value rule (utilize `translate_text` instead)
- removed extraneous badges from README and documentation index

1.3.4 0.0.4 (*2017-10-26*)

- added a better badge provider for PyPi package status
- added support for a `sheet_filter` applied to both value rules and record rules
- added precompilation of filter regexes before application
- added rule application statistics which is now returned from `apply` in a tuple (output_filepath, output_statistics,)
- removed callable filters causing exporting and loading errors (just ignoring callable filters for now)

1.3.5 0.0.3 (*2017-10-25*)

- added more badges to documentation and the README
- fixed (hopefully) the building of documentation for readthedocs.io
- fixed README example with an example that acutally makes sense

1.3.6 0.0.2 (*2017-10-24*)

- added even more badges to the README
- added documentation improvements (linking rules to function references)
- added several small improvements to the tests (better code coverage)

1.3.7 0.0.1 (*2017-10-24*)

- added README formatting fixes for PyPi
- fixed the PyPi configuration issues in setup.py

1.3.8 0.0.0 (*2017-10-24*)

- added the project's base structure (wish i could include change logs for prior structure updates)
- fixed the project's base structure for PY2 compatability

CHAPTER 2

Reference

2.1 sandpaper

This is the base sandpaper package that gets imported.

2.1.1 sandpaper.sandpaper module

`sandpaper.sandpaper.`**`value_rule`**`(`*func*`)`

A meta wrapper for value normalization rules.

> **Note:** Value rules take in a full record and a column name as implicit parameters. They are expected to return the value at `record[column]` that has be normalized by the rule.

Parameters **`func`** (*`callable`*) – The normalization rule

Returns The wrapped normalization rule

Return type callable

`sandpaper.sandpaper.`**`record_rule`**`(`*func*`)`

A meta wrapper for table normalization rules.

> **Note:** Record rules are applied after all value rules have been applied to a record. They take in a full record as an implicit parameter and are expected to return the normalized record back.

Parameters **`func`** (*`callable`*) – The normalization rule

Returns The wrapped normalization rule

Return type callable

class `sandpaper.sandpaper.`**`SandPaper`**(*name=None*)

Bases: `object`

The SandPaper object.

Allows chained data normalization across multiple different table type data files such as `.csv`, `.xls`, and `.xlsx`.

`name`

The descriptive name of the SandPaper instance.

Note: If no name has been given, a continually updating uid hash of the active rules is used instead

Getter Returns the given or suitable name for a SandPaper instance

Setter Sets the descriptive name of the SandPaper instance

Return type str

`uid`

A continually updating hash of the active rules.

A hexadecimal digest string

Getter Returns a continually updating hash of the active rules

Return type str

`rules`

This list of applicable rules for the SandPaper instance.

Getter Returns the list of applicable rules for the instance

Return type list[tuple(callable, tuple(…,…), dict[str,…])]

`value_rules`

The set of value rules for the SandPaper instance.

Getter Returns the set rules for the SandPaper instance

Return type set(callable)

`record_rules`

The set of record rules for the SandPaper instance.

Getter Returns the set rules for the SandPaper instance

Return type set(callable)

`lower`(*record*, *column*, ***kwargs*)

A basic lowercase rule for a given value.

Only applies to text type variables

Parameters

- **`record`** (*`collections.OrderedDict`*) – A record whose value within `column` should be normalized and returned
- **`column`** (*`str`*) – A column that indicates what value to normalize
- **`kwargs`** (*`dict`*) – Any named arguments

Returns The value lowercased

upper(*record*, *column*, ***kwargs*)

A basic uppercase rule for a given value.

Only applies to text type variables

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **kwargs** (*dict*) – Any named arguments

Returns The value uppercased

capitalize(*record*, *column*, ***kwargs*)

A basic capitalization rule for a given value.

Only applies to text type variables

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **kwargs** (*dict*) – Any named arguments

Returns The value capatilized

title(*record*, *column*, ***kwargs*)

A basic titlecase rule for a given value.

Only applies to text type variables

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **kwargs** (*dict*) – Any named arguments

Returns The value titlecased

lstrip(*record*, *column*, *content=None*, ***kwargs*)

A basic lstrip rule for a given value.

Only applies to text type variables.

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **content** (*str*) – The content to strip (defaults to whitespace)
- **kwargs** (*dict*) – Any named arguments

Returns The value with left content stripped

rstrip(*record*, *column*, *content=None*, ***kwargs*)

A basic rstrip rule for a given value.

Only applies to text type variables.

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **content** (*str*) – The content to strip (defaults to whitespace)
- **kwargs** (*dict*) – Any named arguments

Returns The value with right content stripped

strip(*record*, *column*, *content=None*, ***kwargs*)

A basic strip rule for a given value.

Only applies to text type variables.

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **content** (*str*) – The content to strip (defaults to whitespace)
- **kwargs** (*dict*) – Any named arguments

Returns The value with all content stripped

increment(*record*, *column*, *amount=1*, ***kwargs*)

A basic increment rule for a given value.

Only applies to numeric (int, float) type variables.

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **amount** (*int or float*) – The amount to increment by
- **kwargs** (*dict*) – Any named arguments

Returns The value incremented by `amount`

decrement(*record*, *column*, *amount=1*, ***kwargs*)

A basic decrement rule for a given value.

Only applies to numeric (int, float) type variables.

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **column** (*str*) – A column that indicates what value to normalize
- **amount** (*int or float*) – The amount to decrement by
- **kwargs** (*dict*) – Any named arguments

Returns The value incremented by `amount`

`replace`(*record*, *column*, *replacements*, ***kwargs*)

Applies a replacements dictionary to a value.

Take for example the following SandPaper instance:

```
s = SandPaper('my-sandpaper').replace({
    'this_is_going_to_be_replaced': 'with_this',
})
```

Parameters

- **`record`** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **`column`** (*str*) – A column that indicates what value to normalize
- **`replacements`** (*dict[str, str]*) – A dictionary of replacements for the value
- **`kwargs`** (*dict*) – Any named arguments

Returns The value with all replacements made

`translate_text`(*record*, *column*, *translations*, ***kwargs*)

A text translation rule for a given value.

Take for example the following SandPaper instance:

```
s = SandPaper('my-sandpaper').translate_text({
    r'^group(?P<group_id>\d+)\s*(.*)$': '{group_id}'
}, column_filter=r'^group_definition$')
```

This will translate all instances of the value `group<GROUP NUMBER>` to `<GROUP NUMBER>` only in columns named `group_definition`.

Important: Note that matched groups and matched groupdicts are passed as `*args` and `**kwargs` to the format method of the returned `to_format` string.

Parameters

- **`record`** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **`column`** (*str*) – A column that indicates what value to normalize
- **`translations`** (*dict[str, str]*) – A dictionary of translations the value
- **`kwargs`** (*dict*) – Any named arguments

Returns The potentially translated value

`translate_date`(*record*, *column*, *translations*, ***kwargs*)

A date translation rule for a given value.

Take for example the following SandPaper instance:

```
s = SandPaper('my-sandpaper').translate_date({
    '%Y-%m-%d': '%Y',
    '%Y': '%Y',
    '%Y-%m': '%Y'
}, column_filter=r'^(.*)_date$')
```

This will translate all instances of a date value matching the given date formats in columns ending with `_date` to the date format `%Y`.

Parameters

- **`record`** (*`collections.OrderedDict`*) – A record whose value within `column` should be normalized and returned
- **`column`** (*`str`*) – A column that indicates what value to normalize
- **`translations`** (*`dict[str, str]`*) – A dictionary of translations from an arrow based dateformats to a different format
- **`kwargs`** (*`dict`*) – Any named arguments

Returns The value potentially translated value

`add_columns(record, additions, **kwargs)`

Adds columns to a record.

Note: If the value of an entry in `additions` is a callable, then the callable should expect the `record` as the only parameter and should return the value that should be placed in the newly added column.

If the value of an entry in `additions` is a string, the record is passed in as kwargs to the value's `format` method.

Otherwise, the value of an entry in `additions` is simply used as the newly added column's value.

Parameters

- **`record`** (*`collections.OrderedDict`*) – A record whose value within `column` should be normalized and returned
- **`additions`** (*`dict[str, ...]`*) – A dictionary of column names to callables, strings, or other values
- **`kwargs`** (*`dict`*) – Any named arguments

Returns The record with a potential newly added column

`remove_columns(record, removes, **kwargs)`

Removes columns from a record.

Parameters

- **`record`** (*`collections.OrderedDict`*) – A record whose value within `column` should be normalized and returned
- **`removes`** (*`list[str]`*) – A list of columns to remove
- **`kwargs`** (*`dict`*) – Any named arguments

Returns The record with a potential newly removed column

`keep_columns(record, keeps, **kwargs)`

Removes all other columns from a record.

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **keeps** (*list[str]*) – A list of columns to keep
- **kwargs** (*dict*) – Any named arguments

Returns The record with a potential newly kept column

rename_columns (*record*, *renames*, ***kwargs*)

Maps an existing column to a new column.

Parameters

- **record** (*collections.OrderedDict*) – A record whose value within `column` should be normalized and returned
- **renames** (*dict[str, str]*) – A dictionary of column to column renames
- **kwargs** (*dict*) – Any named arguments

Returns The record with the remapped column

order_columns (*record*, *order*, *ignore_missing=False*, ***kwargs*)

Orders columns in a specific order.

Parameters

- **record** (*collections.OrderedDict*) – A record who should be ordered
- **order** (*list[str]*) – The order that columns need to be in
- **ignore_missing** (*bool*) – Boolean which inidicates if missing columns from `order` should be ignored
- **kwargs** (*dict*) – Any named arguments

Returns The record with the columns reordered

apply (*from_file*, *to_file*, *sheet_name=None*, *row_filter=None*, *monitor_rules=False*, ***kwargs*)

Applies a SandPaper instance rules to a given glob of files.

Parameters

- **from_file** (*str*) – The path of the file to apply the rules to
- **to_file** (*str*) – The path of the file to write to
- **sheet_name** (*str*) – The name of the sheet to apply rules to (defaults to the first available sheet)
- **row_filter** (*callable*) – A callable which accepts a cleaned record and returns True if the record should be written out
- **monitor_rules** (*bool*) – Boolean flag that inidicates if the count of applied rules should be monitored
- **kwargs** (*dict*) – Any additional named arguments (applied to the pyexcel `iget_records` method)

Returns The rule statistics if `monitor_rules` is true

Return type dict[str, int]

classmethod from_json (*serialization*)

Loads a SandPaper instance from a json serialization.

Note: Raises a `UserWarning` when the loaded instance does not match the serialized instance's `uid`.

Parameters **`serialization`** (*`dict`*) – The read json serialization

Returns A new SandPaper instance

Return type *SandPaper*

CHAPTER 3

Indices

- genindex
- modindex
- search

Python Module Index

Index