Table Enforcer Documentation

Release 0.4.4

Gus Dunn

Feb 15, 2018

Contents

1 Table Enforcer **3**
- 1.1 Demo Usage . . . 3
- 1.2 Description . . . 3
- 1.3 Basic Workflow . . . 3
- 1.4 Features . . . 4
- 1.5 Credits . . . 4

2 Installation **5**
- 2.1 From sources . . . 5

3 Usage **7**

4 Source Code Documentation **9**
- 4.1 table_enforcer package . . . 9

5 Contributing **15**
- 5.1 Types of Contributions . . . 15
- 5.2 Get Started! . . . 16
- 5.3 Pull Request Guidelines . . . 17
- 5.4 Tips . . . 17

6 Credits **19**
- 6.1 Development Lead . . . 19
- 6.2 Contributors . . . 19

7 History **21**
- 7.1 v0.4.4 / 2018-02-15 . . . 21
- 7.2 v0.4.3 / 2018-02-15 . . . 21
- 7.3 v0.4.2 / 2018-02-15 . . . 21
- 7.4 v0.4.1 / 2018-02-14 . . . 21
- 7.5 v0.4.0 / 2018-02-13 . . . 22
- 7.6 v0.3.0 / 2018-02-07 . . . 22
- 7.7 v0.2.0 / 2018-02-02 . . . 23
- 7.8 v0.1.5 / 2018-02-01 . . . 23
- 7.9 v0.1.4 / 2018-01-26 . . . 23
- 7.10 v0.1.3 / 2018-01-26 . . . 23
- 7.11 v0.1.2 / 2017-11-17 . . . 23

7.12 v0.1.1 / 2017-11-16 24
7.13 v0.1.0 / 2017-11-15 24
7.14 v0.0.1 / 2017-11-14 24

8 Indices and tables **25**

Python Module Index **27**

Contents:

CHAPTER 1

Table Enforcer

1.1 Demo Usage

Have a look at this Demo Notebook

1.2 Description

A python package to facilitate the iterative process of developing and using schema-like representations of table data to recode and validate instances of these data stored in pandas DataFrames. This is a *fairly young* attempt to solve a recurrent problem many people have. So far I have looked at multiple solutions, but none really did it for me.

They either deal primarily with JSON encoded data or they only really solve the validation side of the problem and consider recoding to be a separate issue. They seem to assume that recoding and cleaning has already been done and all we care about is making sure the final product is sane.

To me, this seems backwards.

I need to load, recode, and validate tables all day, everyday. Sometimes its simple; I can `pandas.read_table()` and all is good. But sometimes I have a 700 column long RedCap data dump that is complicated af, and it *really* helps me to develop my recoding logic through an iterative process. For me it makes sense to couple the recoding process directly with the validation process: to write the "tests" for each column first, then add recoding logic in steps until the tests pass.

So *Table Enforcer* is my attempt to apply a sort of "test driven development" workflow to data cleaning and validation.

1.3 Basic Workflow

1. For each column that you care about in your source table:

   (a) Define a `Column` object that represents the ideal state of your data by passing a list of small, independent, reusable validator functions and some descriptive information.
   (b) Use this object to validate the column data from your source table.
      - It will probably fail.
   (c) Add small, composable, reusable recoding functions to the column object and iterate until your validations pass.
2. Define an `Enforcer` object by passing it a list of your column representation objects.
3. This enforcer can be used to recode or validate recoded tables of the same kind as your source table wherever your applications use that type of data.

Please take a look and offer thoughts/advice.

- Free software: MIT license
- Web site: https://github.com/xguse/table_enforcer
- Documentation: https://table-enforcer.readthedocs.io.

1.4 Features

- `Enforcer` and `Column` classes to define what columns should look like in a table.
- `CompundColumn` class that supports complex operations including "one-to-many" and "many-to-one" recoding logic as sometimes a column tries to do too much and should really be multiple columns as well as the reverse.
- Growing cadre of built-in validator functions and decorators.
- Decorators for use in defining parameterized validators like `between_4_and_60()`.

1.5 Credits

This package was created with Cookiecutter and the xguse/cookiecutter-pypackage project template which is based on audreyr/cookiecutter-pypackage.

CHAPTER 2

Installation

2.1 From sources

The sources for Table Enforcer can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/xguse/table_enforcer
```

Or download the tarball:

```
$ curl  -OL https://github.com/xguse/table_enforcer/tarball/master
```

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

1. Navigate to the main repository directory.
2. Activate whichever virtual environment that you want the package installed to.
3. Run the following command.

```
$ pip install .
```

CHAPTER 3

Usage

Have a look at this Demo Notebook

CHAPTER 4

Source Code Documentation

4.1 table_enforcer package

4.1.1 Subpackages

table_enforcer.utils package

Subpackages

table_enforcer.utils.recode package

Submodules

table_enforcer.utils.recode.funcs module

Provide builtin recoder functions for common use cases.

Like validators, recoders take a single *pandas.Series* object as input and return a *pandas.Series* of the same shape and indexes as the original series object. However, instead of returning a series of *True*/*False* values, it performs some operation on the data that gets the column data closer to being how you want it to look during analysis operations.

`table_enforcer.utils.recode.funcs.`**`lower`**(*series*)

Transform all text to lowercase.

`table_enforcer.utils.recode.funcs.`**`upper`**(*series*)

Transform all text to uppercase.

Module contents

table_enforcer.utils.validate package

Submodules

table_enforcer.utils.validate.decorators module

Provide decoration functions to augment the behavior of validator functions.

`table_enforcer.utils.validate.decorators.`**`bounded_length`**(*low*, *high=None*)
Test that the length of the data items fall within range: low <= x <= high.

If high is None, treat as exact length.

`table_enforcer.utils.validate.decorators.`**`choice`**(*choices*)
Test that the data items are members of the set *choices*.

`table_enforcer.utils.validate.decorators.`**`minmax`**(*low*, *high*)
Test that the data items fall within range: low <= x <= high.

table_enforcer.utils.validate.funcs module

Provide builtin validator functions for common use cases.

In general, validators take a single *pandas.Series* object as input and return a *pandas.Series* of the same shape and indexes containing *True* or *False* relative to which items passed the validation logic.

`table_enforcer.utils.validate.funcs.`**`lower`**(*series*)
Test that the data items are all lowercase.

`table_enforcer.utils.validate.funcs.`**`negative`**(*series: pandas.core.series.Series*) → pandas.core.series.Series
Test that the data items are negative.

`table_enforcer.utils.validate.funcs.`**`not_null`**(*series: pandas.core.series.Series*) → pandas.core.series.Series
Return Series with True/False bools based on which items pass.

`table_enforcer.utils.validate.funcs.`**`positive`**(*series: pandas.core.series.Series*) → pandas.core.series.Series
Test that the data items are positive.

`table_enforcer.utils.validate.funcs.`**`unique`**(*series: pandas.core.series.Series*) → pandas.core.series.Series
Test that the data items do not repeat.

`table_enforcer.utils.validate.funcs.`**`upper`**(*series*)
Test that the data items are all uppercase.

Module contents

Module contents

4.1.2 Submodules

4.1.3 table_enforcer.errors module

Provide error classes.

exception table_enforcer.errors.**NotImplementedYet**(*msg=None*)

Bases: NotImplementedError, *table_enforcer.errors.TableEnforcerError*

Raise when a section of code that has been left for another time is asked to execute.

__init__(*msg=None*)

Set up the Exception.

exception table_enforcer.errors.**RecodingError**(*column*, *recoder*, *exception*)

Bases: *table_enforcer.errors.TableEnforcerError*

Raise when a recoder function raises an error.

__init__(*column*, *recoder*, *exception*)

Set up the Exception.

exception table_enforcer.errors.**TableEnforcerError**

Bases: Exception

Base error class.

exception table_enforcer.errors.**ValidationError**

Bases: *table_enforcer.errors.TableEnforcerError*

Raise when a validator function fails to generate all successes when called inside of a *recode* method.

4.1.4 table_enforcer.main_classes module

Main module.

class table_enforcer.main_classes.**Enforcer**(*columns*)

Bases: object

Class to define table definitions.

__init__(*columns*)

Initialize an enforcer instance.

_make_validations(*table: pandas.core.frame.DataFrame*) → box.Box

Return a dict-like object containing dataframes of which tests passed/failed for each column.

recode(*table: pandas.core.frame.DataFrame*, *validate=False*) → pandas.core.frame.DataFrame

Return a fully recoded dataframe.

Parameters

- **table** (*pd.DataFrame*) – A dataframe on which to apply recoding logic.
- **validate** (*bool*) – If True, recoded table must pass validation tests.

validate(*table: pandas.core.frame.DataFrame*) → bool
Return True if all validation tests pass: False otherwise.

class table_enforcer.main_classes.**BaseColumn**
Bases: `object`

Base Class for Columns.

Lays out essential methods api.

recode(*table: pandas.core.frame.DataFrame*, *validate=False*) → pandas.core.frame.DataFrame
Pass the appropriate columns through each recoder function sequentially and return the final result.

Parameters

- **table** (*pd.DataFrame*) – A dataframe on which to apply recoding logic.
- **validate** (*bool*) – If `True`, recoded table must pass validation tests.

update_dataframe(*df*, *table*, *validate=False*)
Perform `self.recode` and add resulting column(s) to `df` and return `df`.

validate(*table: pandas.core.frame.DataFrame*, *failed_only=False*) → pandas.core.frame.DataFrame
Return a dataframe of validation results for the appropriate series vs the vector of validators.

Parameters

- **table** (*pd.DataFrame*) – A dataframe on which to apply validation logic.
- **failed_only** (*bool*) – If `True`: return only the indexes that failed to validate.

class table_enforcer.main_classes.**Column**(*name: str, dtype: type, unique: bool, validators: typing.List[typing.Callable[pandas.core.series.Series, pandas.core.frame.DataFrame]], recoders: typing.List[typing.Callable[pandas.core.series.Series, pandas.core.series.Series]]*) → None
Bases: *table_enforcer.main_classes.BaseColumn*

Class representing a single table column.

__init__(*name: str, dtype: type, unique: bool, validators: typing.List[typing.Callable[pandas.core.series.Series, pandas.core.frame.DataFrame]], recoders: typing.List[typing.Callable[pandas.core.series.Series, pandas.core.series.Series]]*) → None
Construct a new *Column* object.

Parameters

- **name** (*str*) – The exact name of the column in a `pd.DataFrame`.
- **dtype** (*type*) – The type that each member of the recoded column must belong to.
- **unique** (*bool*) – Whether values are allowed to recur in this column.
- **validators** (*list*) – A list of validator functions.
- **recoders** (*list*) – A list of recoder functions.

_dict_of_funcs(*funcs: list*) → pandas.core.series.Series
Return a pd.Series of functions with index derived from the function name.

_validate_series_dtype(*series: pandas.core.series.Series*) → pandas.core.series.Series
Validate that the series data is the correct dtype.

recode(*table: pandas.core.frame.DataFrame*, *validate=False*) → pandas.core.frame.DataFrame
Pass the provided series obj through each recoder function sequentially and return the final result.

Parameters

- **table** (*pd.DataFrame*) – A dataframe on which to apply recoding logic.
- **validate** (*bool*) – If `True`, recoded table must pass validation tests.

validate(*table: pandas.core.frame.DataFrame*, *failed_only=False*) → pandas.core.frame.DataFrame
Return a dataframe of validation results for the appropriate series vs the vector of validators.

Parameters

- **table** (*pd.DataFrame*) – A dataframe on which to apply validation logic.
- **failed_only** (*bool*) – If `True`: return only the indexes that failed to validate.

class table_enforcer.main_classes.**CompoundColumn**(*input_columns: typing.List[table_enforcer.main_classes.Column], output_columns: typing.List[table_enforcer.main_classes.Column], column_transform*) → None

Bases: *table_enforcer.main_classes.BaseColumn*

Class representing multiple columns and the logic governing their transformation from source table to recoded table.

__init__(*input_columns: typing.List[table_enforcer.main_classes.Column], output_columns: typing.List[table_enforcer.main_classes.Column], column_transform*) → None
Construct a new `CompoundColumn` object.

Parameters

- **input_columns** (*list, Column*) – A list of `Column` objects representing column(s) from the SOURCE table.
- **output_columns** (*list, Column*) – A list of `Column` objects representing column(s) from the FINAL table.
- **column_transform** (*Callable*) – Function accepting the table object, performing transformations to it and returning a DataFrame containing the NEW columns only.

_do_validation_set(*table: pandas.core.frame.DataFrame*, *columns*, *validation_type*, *failed_only=False*) → pandas.core.frame.DataFrame
Return a dataframe of validation results for the appropriate series vs the vector of validators.

_validate_input(*table: pandas.core.frame.DataFrame*, *failed_only=False*) → pandas.core.frame.DataFrame
Return a dataframe of validation results for the appropriate series vs the vector of validators.

recode(*table: pandas.core.frame.DataFrame*, *validate=False*) → pandas.core.frame.DataFrame
Pass the appropriate columns through each recoder function sequentially and return the final result.

Parameters

- **table** (*pd.DataFrame*) – A dataframe on which to apply recoding logic.
- **validate** (*bool*) – If `True`, recoded table must pass validation tests.

validate(*table: pandas.core.frame.DataFrame*, *failed_only=False*) → pandas.core.frame.DataFrame
Return a dataframe of validation results for the appropriate series vs the vector of validators.

Parameters

- **table** (*pd.DataFrame*) – A dataframe on which to apply validation logic.

- **failed_only** (*bool*) – If `True`: return only the indexes that failed to validate.

4.1.5 Module contents

Top-level package for Table Enforcer.

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/xguse/table_enforcer/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/xguse/table_enforcer/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 :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/table_enforcer.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 table_enforcer
$ cd table_enforcer/
$ 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, including testing other Python versions with tox:

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

   To get flake8 and tox, just pip install them 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.

5.3 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 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/xguse/table_enforcer/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_table_enforcer
```

CHAPTER 6

Credits

6.1 Development Lead

- Gus Dunn <w.gus.dunn@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 v0.4.4 / 2018-02-15

- fix recode/validate access to sub-pkgs
- ignore .pytest_cache

7.2 v0.4.3 / 2018-02-15

- Fixed import errors
- ignore test_chamber

7.3 v0.4.2 / 2018-02-15

- Address import errors when not installed editable
- update README link to Usage_Demo
- ship docs/_static/Usage_Demo.html
- Updated Usage_Demo
- added to doctrings in main_classes

7.4 v0.4.1 / 2018-02-14

- added readthedocs.yml
- Updated Usage_Demo and README

7.5 v0.4.0 / 2018-02-13

- Updated tests for CompoundColumn
- CompoundColumn absorbs MTO/OTM-subclasses
- updated tests/files/demo_table*.csv
- updated docs/demo_notebook
- OTMColumn.input_columns must be len == 1
- amended tests for new OTMColumn
- main_classes: rewrite OTMColumn and general reorg
- BaseColumn method defs now sets api for subclasses
- Enforcer.columns is now simple list
- setup.cfg: whitelist varname df
- main_classes: restruct base classes + ComplexColumn
- main_classes: col takes table
- test_column: col takes table
- add testing files for MTOColumn
- ignore LibreOffice lock files
- OTMColumn: improved __doc__
- update_dataframe: call sig now has *validate*

7.6 v0.3.0 / 2018-02-07

- main_classes: OTMColumn is functional
- updated testing for OTMColumn
- main_classes: replace Munch w/ Box (probationary)
- add python-box to reqs (probationary)
- conftest: modularize paths
- add testing for OTMColumn
- test_column: fix typos and style
- import all from main_classes
- Bump version: 0.1.5 → 0.2.0
- changelog(v0.2.0)
- Updated Docs version Usage_Demo.ipynb

7.7 v0.2.0 / 2018-02-02

- Enforcer.recode lets Column.recode do the validation now
- Enforcer.validate no longer recodes
- Enforcer: make_validations now private
- Column: added find_failed_rows()
- columns now take series not dataframe
- added system-lvl tests based on Usage_Demo.ipynb
- Enforcer.recode create new df rather than copy
- added RecoderError and focused ValidationError
- remove testing for 3.5
- dont lint tests
- ignore flake8:W292
- formatting

7.8 v0.1.5 / 2018-02-01

- Added tests for imports and more Class behavior
- main_classes: calling recode with validate is now prefered

7.9 v0.1.4 / 2018-01-26

- main_classes.py: removed faulty imports

7.10 v0.1.3 / 2018-01-26

- corrected Usage_Demo.ipynb
- formatting and typing
- table_enforcer.py -> main_classes.py

7.11 v0.1.2 / 2017-11-17

- flake8
- set up basic testing
- changed travis build settings
- updated usage demo and readme

7.12 v0.1.1 / 2017-11-16

- Added usage notebook link to docs.
- reorganized import strategy of Enforcer/Column objs
- added more builtin validators/recoders/decorators
- updated reqs
- initialized travis integration
- updated docs
- Added usage demo notebook for docs
- updated ignore patterns
- validators.py: renamed

7.13 v0.1.0 / 2017-11-15

- first minimally functional package
- Enforcer and Column classes defined and operational
- small cadre of built-in validator functions and decorators
- ignore jupyter stuff
- linter setups

7.14 v0.0.1 / 2017-11-14

- First commit

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

table_enforcer, 14
table_enforcer.errors, 11
table_enforcer.main_classes, 11
table_enforcer.utils, 11
table_enforcer.utils.recode, 10
table_enforcer.utils.recode.funcs, 9
table_enforcer.utils.validate, 11
table_enforcer.utils.validate.decorators, 10
table_enforcer.utils.validate.funcs, 10

Index

Symbols

__init__() (table_enforcer.errors.NotImplementedYet method), 11
__init__() (table_enforcer.errors.RecodingError method), 11
__init__() (table_enforcer.main_classes.Column method), 12
__init__() (table_enforcer.main_classes.CompoundColumn method), 13
__init__() (table_enforcer.main_classes.Enforcer method), 11
_dict_of_funcs() (table_enforcer.main_classes.Column method), 12
_do_validation_set() (table_enforcer.main_classes.CompoundColumn method), 13
_make_validations() (table_enforcer.main_classes.Enforcer method), 11
_validate_input() (table_enforcer.main_classes.CompoundColumn method), 13
_validate_series_dtype() (table_enforcer.main_classes.Column method), 12

B

BaseColumn (class in table_enforcer.main_classes), 12
bounded_length() (in module table_enforcer.utils.validate.decorators), 10

C

choice() (in module table_enforcer.utils.validate.decorators), 10
Column (class in table_enforcer.main_classes), 12
CompoundColumn (class in table_enforcer.main_classes), 13

E

Enforcer (class in table_enforcer.main_classes), 11

L

lower() (in module table_enforcer.utils.recode.funcs), 9
lower() (in module table_enforcer.utils.validate.funcs), 10

M

minmax() (in module table_enforcer.utils.validate.decorators), 10

N

negative() (in module table_enforcer.utils.validate.funcs), 10
not_null() (in module table_enforcer.utils.validate.funcs), 10
NotImplementedYet, 11

P

positive() (in module table_enforcer.utils.validate.funcs), 10

R

recode() (table_enforcer.main_classes.BaseColumn method), 12
recode() (table_enforcer.main_classes.Column method), 12
recode() (table_enforcer.main_classes.CompoundColumn method), 13
recode() (table_enforcer.main_classes.Enforcer method), 11
RecodingError, 11

T

table_enforcer (module), 14
table_enforcer.errors (module), 11
table_enforcer.main_classes (module), 11
table_enforcer.utils (module), 11
table_enforcer.utils.recode (module), 10
table_enforcer.utils.recode.funcs (module), 9
table_enforcer.utils.validate (module), 11
table_enforcer.utils.validate.decorators (module), 10

table_enforcer.utils.validate.funcs (module), 10
TableEnforcerError, 11

U

unique() (in module table_enforcer.utils.validate.funcs), 10
update_dataframe() (table_enforcer.main_classes.BaseColumn method), 12
upper() (in module table_enforcer.utils.recode.funcs), 9
upper() (in module table_enforcer.utils.validate.funcs), 10

V

validate() (table_enforcer.main_classes.BaseColumn method), 12
validate() (table_enforcer.main_classes.Column method), 13
validate() (table_enforcer.main_classes.CompoundColumn method), 13
validate() (table_enforcer.main_classes.Enforcer method), 11
ValidationError, 11