pysemantic Documentation

Release 0.0.1

Jaidev Deshpande

July 01, 2015

Contents

Contents:

CHAPTER 1

Examples

- Introduction to PySemantic.

CHAPTER 2

Schema Configuration Reference

Every project in PySemantic can be configured via a data dictionary or a schema, which is a yaml file. This file houses the details of how PySemantic should treat a project's constituent datasets. A typical data dictionary follows the following pattern:

```
dataset_name:
  dataset_param_1: value1
  dataset_param_2: value2
  # etc
```

PySemantic reads this as a dictionary where the parameter names are keys and their values are the values in the dictionary. Thus, the schema for a whole project is a dictionary of dictionaries.

2.1 Basic Schema Configuration

Here is a list of different dataset parameters that PySemantic is sensitive to:

- `path` (Required) The absolute path to the file containing the data. Note that the path must be absolute. This can also be a list of files if the dataset spans multiple files. If that is the case, the path parameter can be specified as:

```
    path:
      - absolulte/path/to/file/1
      - absolulte/path/to/file/2
      # etc
```

- `demlimiter` (Optional, default: `,`) The delimiter used in the file. This has to be a character delimiter, not words like "comma" or "tab".
- `md5` (Optional) The MD5 checksum of the file to read. This necessary because sometimes we read files and after processing it, rewrite to the same path. This parameter helps keep track of whether the file is correct.
- `header`: (Optional) The header row of the file.
- `index_col`: (Optional) Name of the column that forms the index of the dataframe. This can be a single string or a list of strings. If a list is provided, the dataframe becomes multi-indexed.
- `sheetname`: (Optional) Name of the sheet containing the dataset in an MS Excel spreadsheet. This comes into play only when `path` points to an Excel file. For other types of files, this is ignored. When `path` is an Excel file and this parameter is not provided, it is assumed to be the same as the name of the dataset. For example:

```
    iris:
        path: /path/to/iris.xlsx
```

The schema above assumes that the iris dataset resides in a sheet named "iris". If instead the name of the sheet is different, you can specify it as:

```
iris:
    path: /path/to/iris.xlsx
    sheetname: name_of_sheet
```

This parameter can also be a list, to enable the combination of multiple sheets into a dataframe, as follows:

```
iris:
    path: /path/to/iris.xlsx
    sheetname:
        - sheet1
        - sheet2
```

This will combine the data from sheet1 and sheet2 into a single dataframe.

- `column_names`: (Optional) Specify the names of columns to use in the loaded dataframe. This option can have multiple types of values. It can be:

    1. A list of strings to use as column names:

```
column_names:
  - column_1
  - column_2
  - column_3
```

    2. A dictionary that maps original column names to new ones:

```
column_names:
  org_colname_1: new_colname_a
  org_colname_2: new_colname_b
  org_colname_3: new_colname_c
```

    3. A Python function that translates the name of every column in the loaded dataframe:

```
column_names: !!python/name:module_name.translate_column_name
```

- `nrows`: (Optional) Method to select which rows are read from the dataset. This option, like `column_names`, can be specified in many ways. It can be:

    1. An integer (default): Number of rows to read from the file. If this option is not specified, all rows from the file are read.

```
nrows: 100
```

    2. A dictionary that recognizes specific keys:

        – `random`: A boolean that directs PySemantic to shuffle the selected rows after loading the dataset. For example, including the following lines in the schema

```
nrows:
   random: true
```

        will shuffle the dataset before returning it.

        – `range`: A list of two integers, which denote the first and the last index of the range of rows to be read. For example, the following lines

```
nrows:
    range:
        - 10
        - 50
```

will only select the 10th to the 50th (exclusive) rows.

- `count`: An integer that can be used in conjunction with either or both of the above options, to denote the number of rows to read from a random selection or a range.

```
nrows:
    range:
        - 10
        - 50
    count: 10
    random: true
```

The lines shown above will direct PySemantic to load 10 rows at random between the 10th and the 50th rows of a dataset.

3. A callable which returns a logical array which has the same number of elements as the number of rows in the dataset. The output of this callable is used as a logical index for slicing the dataset. For example, suppose we wanted to extract all even numbered rows from a dataset, then we could make a callable as follows:

```
iseven = lambda x: np.remainder(x, 2) == 0
```

Suppose this function resides in a module called `foo.bar`, then we can include it in the schema as follows:

```
nrows: !!python/name:foo.bar.iseven
```

This will cause PySemantic to only load all even valued row numbers.

- `use_columns`: (Optional) The list of the columns to read from the dataset. The format for specifying this parameter is as follows:

```
use_columns:
  - column_1
  - column_2
  - column_3
```

If this parameter is not specified, all columns present in the dataset are read.

- `exclude_columns`: This option can be used to specify columns that are explicityly to be ignored. This is useful when there are large number of columns in the dataset and we only wish to exclude a few. Note that this option overrides the `use_columns` option, i.e. if a column name is present in both lists, it will be dropped.
- `na_values`: A string or a list of values that are considered as NAs by the pandas parsers, applicable to the whole dataframe.
- `converters`: A dictionary of functions to be applied to columns when loading data. Any Python callable can be added to this list. This parameter makes up the `converters` argument of Pandas parsers. The usage is as follows:

```
converters:
  col_a: !!python/name:numpy.int
```

This results in the `numpy.int` function being called on the column `col_a`

- `dtypes` (Optional) Data types of the columns to be read. Since types in Python are native objects, PySemantic expects them to be so in the schema. This can be formatted as follows:

```
dtypes:
  column_name: !!python/name:python_object
```

For example, if you have three columns named `foo`, `bar`, and `baz`, which have the types `string`, `integer` and `float` respectively, then your schema should look like:

```
dtypes:
  foo: !!python/name:__builtin__.str
  bar: !!python/name:__builtin__.int
  baz: !!python/name:__builtin__.float
```

Non-builtin types can be specified too:

```
dtypes:
  datetime_column: !!python/name:datetime.date
```

Note: You can figure out the yaml representation of a Python type by doing the following:

```
import yaml
x = type(foo) # where foo is the object who's type is to be yamlized
print yaml.dump(x)
```

- `combine_dt_columns` (Optional) Columns containing Date/Time values can be combined into one column by using the following schema:

```
combine_dt_columns:
  output_col_name:
    - col_a
    - col_b
```

This will parse columns `col_a` and `col_b` as datetime columns, and put the result in a column named `output_col_name`. Specifying the output name is optional. You may declare the schema as:

```
combine_dt_columns:
  - col_a
  - col_b
```

In this case the parser will simply name the output column as `col_a_col_b`, as is the default with Pandas.

NOTE: Specifying this column will make PySemantic ignore any columns that have been declared as having the datetime type in the `dtypes` parameter.

- `pickle` (Optional) Absolute path to file which contains pickled arguments for the parser. This option can be used if readability or declaratives are not a concern. The file should contain a picked dictionary that is directly passed to the parser, i.e. if the loaded pickled data is in a dict named `data`, then parser invocation becomes `parser(**data)`.

NOTE: If any of the above options are present, they will override the corresponding arguments contained in the pickle file. In PySemantic, declarative statements have the right of way.

2.2 Column Schema Configuration

PySemantic also allows specifying rules and validators independently for each column. This can be done using the `column_rules` parameter of the dataset schema. Here is a typical format:

```
dataset_name:
  column_rules:
    column_1_name:
      # rules to be applied to the column
    column_2_name:
      # rules to be applied to the column
```

The following parameters can be supplied to any column under `column_rules`:

- `is_drop_na` ([true|false], default false) Setting this to `true` causes PySemantic to drop all NA values in the column.
- `is_drop_duplicates` ([true|false], default false) Setting this to `true` causes PySemantic to drop all duplicated values in the column.
- `unique_values`: These are the unique values that are expected in a column. The value of this parameter has to be a yaml list. Any value not found in this list will be dropped when cleaning the dataset.
- `exclude`: These are the values that are to be explicitly excluded from the column. This comes in handy when a column has too many unique values, and a handful of them have to be dropped.
- `minimum`: Minimum value allowed in a column if the column holds numerical data. By default, the minimum is -np.inf. Any value less than this one is dropped.
- `maximum`: Maximum value allowed in a column if the column holds numerical data. By default, the maximum is np.inf. Any value greater than this one is dropped.
- `regex`: A regular expression that each element of the column must match, if the column holds text data. Any element of the column not matching this regex is dropped.
- `na_values`: A list of values that are considered as NAs by the pandas parsers, applicable to this column.
- `postprocessors`: A list of callables that called one by one on the columns. Any python function that accepts a series, and returns a series can be a postprocessor.

Here is a more extensive example of the usage of this schema.

```
iris:
  path: /home/username/src/pysemantic/testdata/iris.csv
  converters:
    Sepal Width: !!python/name:numpy.floor
  column_rules:
    Sepal Length:
      minimum: 2.0
    Petal Length:
      maximum: 4.0
    Petal Width:
      exclude:
        - 3.14
    Species:
      unique_values:
        - setosa
        - versicolor
      postprocessors:
        - !!python/name:module_name.foo
```

This would cause PySemantic to produce a dataframe corresponding to the Fisher iris dataset which has the following characteristics:

1. It contains no observations where the sepal length is less than 2 cm.
2. It contains no observations where the petal length is more than 4 cm.

3. The sepal width only contains integers.
4. The petal width column will not contain the specific value 3.14
5. The species column will only contain the values "setosa" and "versicolor", i.e. it will not contain the value "virginica".
6. The species column in the dataframe will be processed by the `module_name.foo` function.

2.3 DataFrame Schema Configuration

A few rules can also be enforced at the dataframe level, instead of at the level of individual columns in the dataset. Two of them are:

- `drop_duplicates` ([true|false, default true]). This behaves in the same way as `is_drop_duplicates` for series schema, with the exception that here the default is True.
- `drop_na` ([true|false, default true]). This behaves in the same way as `is_drop_na` for series schema, with the exception that here the default is True.

CHAPTER 3

API Reference

3.1 pysemantic package

3.1.1 Submodules

3.1.2 pysemantic.cli module

semantic

Usage: semantic list [–project=<PROJECT_NAME>] semantic add PROJECT_NAME PROJECT_SPECFILE semantic remove PROJECT_NAME [–dataset=<dname>] semantic set-schema PROJECT_NAME SCHEMA_FPATH semantic set-specs PROJECT_NAME –dataset=<dname> [–path=<pth>] [–dlm=<sep>] semantic add-dataset DATASET_NAME –project=<pname> –path=<pth> –dlm=<sep> semantic export PROJECT_NAME [–dataset=<dname>] OUTPATH

Options: -h –help Show this screen -d –dataset=<dname> Name of the dataset to modify –path=<pth> Path to a dataset –dlm=<sep> Declare the delimiter for a dataset -p –project=<pname> Name of the project to modify -v –version Print the version of PySemantic

pysemantic.cli.**cli**(*arguments*)

cli - The main CLI argument parser.

Parameters **arguments** (*dict*) – command line arguments, as parsed by docopt

Returns None

pysemantic.cli.**main**()

3.1.3 pysemantic.custom_traits module

Customized traits for advanced validation.

class pysemantic.custom_traits.**AbsFile**(*value=''*, *filter=None*, *auto_set=False*, *entries=0*, *exists=False*, ***metadata*)

Bases: traits.trait_types.File

A File trait whose value must be an absolute path, to an existing file.

validate(*obj*, *name*, *value*)

class pysemantic.custom_traits.**DTypeTraitDictObject**(*trait*, *object*, *name*, *value*)

Bases: traits.trait_handlers.TraitDictObject

Subclassed from the parent to aid the validation of DTypesDicts.

class pysemantic.custom_traits.**DTypesDict**(*key_trait=None*, *value_trait=None*, *value=None*, *items=True*, ***metadata*)

Bases: traits.trait_types.Dict

A trait whose keys are strings, and values are Type traits. Ideally this is the kind of dictionary that is passed as the *dtypes* argument in *pandas.read_table*.

validate(*obj*, *name*, *value*)

Subclassed from the parent to return a *DTypeTraitDictObject* instead of *traits.trait_handlers.TraitDictObhject*.

class pysemantic.custom_traits.**NaturalNumber**(*default_value=<traits.trait_handlers.NoDefaultSpecified object>*, ***metadata*)

Bases: traits.trait_types.BaseInt

An integer trait whose value is a natural number.

default_value = 1

error(*obj*, *name*, *value*)

validate(*obj*, *name*, *value*)

class pysemantic.custom_traits.**ValidTraitList**(*trait=None*, *value=None*, *minlen=0*, *maxlen=9223372036854775807*, *items=True*, ***metadata*)

Bases: traits.trait_types.List

A List trait whose every element should be valid trait.

validate(*obj*, *name*, *value*)

3.1.4 pysemantic.errors module

Errors.

exception pysemantic.errors.**MissingConfigError**

Bases: exceptions.Exception

Error raised when the pysemantic configuration file is not found.

exception pysemantic.errors.**MissingProject**

Bases: exceptions.Exception

Error raised when project is not found.

3.1.5 pysemantic.exporters module

Exporters from PySemantic to databases or other data sinks.

class pysemantic.exporters.**AbstractExporter**

Bases: object

Abstract exporter for dataframes that have been cleaned.

get(***kwargs*)

set(***kwargs*)

class pysemantic.exporters.**AerospikeExporter**(*config*, *dataframe*)

Bases: *pysemantic.exporters.AbstractExporter*

Example class for exporting to an aerospike database.

run()

set(*key_tuple*, *bins*)

3.1.6 pysemantic.loggers module

Loggers

pysemantic.loggers.**setup_logging**(*project_name*)

3.1.7 pysemantic.project module

The Project class.

class pysemantic.project.**Project**(*project_name=None*, *parser=None*, *schema=None*)
Bases: object

The Project class, the entry point for most things in this module.

datasets
"List the datasets registered under the parent project.

Example

```
>>> project = Project('skynet')
>>> project.datasets
['sarah connor', 'john connor', 'kyle reese']
```

export_dataset(*dataset_name*, *dataframe=None*, *outpath=None*)
Export a dataset to an exporter defined in the schema. If nothing is specified in the schema, simply export to a CSV file such named <dataset_name>.csv

Parameters

- **dataset_name** (*Str*) – Name of the dataset to exporter.
- **dataframe** – Pandas dataframe to export. If None (default), this dataframe is loaded using the *load_dataset* method.

get_dataset_specs(*dataset_name*)
Returns the specifications for the specified dataset in the project.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns Parser arguments required to import the dataset in pandas.

Return type dict

get_project_specs()
Returns a dictionary containing the schema for all datasets listed under this project.

Returns Parser arguments for all datasets listed under the project.

Return type dict

load_dataset(*dataset_name*)
Load and return a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns A pandas DataFrame containing the dataset.

Return type pandas.DataFrame

Example

```
>>> demo_project = Project('pysemantic_demo')
>>> iris = demo_project.load_dataset('iris')
>>> type(iris)
pandas.core.DataFrame
```

load_datasets()
Load and return all datasets.

Returns dictionary like {dataset_name: dataframe}

Return type dict

reload_data_dict()
Reload the data dictionary and re-populate the schema.

set_dataset_specs(*dataset_name*, *specs*, *write_to_file=False*)
Sets the specifications to the dataset. Using this is not recommended. All specifications for datasets should be handled through the data dictionary.

Parameters

- **dataset_name** (*str*) – Name of the dataset for which specifications need to be modified.
- **specs** (*dict*) – A dictionary containing the new specifications for the dataset.
- **write_to_file** (*bool*) – If true, the data dictionary will be updated to the new specifications. If False (the default), the new specifications are used for the respective dataset only for the lifetime of the *Project* object.

Returns None

update_dataset(*dataset_name*, *dataframe*, *path=None*, ***kwargs*)
This is tricky.

view_dataset_specs(*dataset_name*)
Pretty print the specifications for a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

pysemantic.project.**add_dataset**(*project_name*, *dataset_name*, *dataset_specs*)
Add a dataset to a project.

Parameters

- **project_name** (*str*) – Name of the project to which the dataset is to be added.
- **dataset_name** (*str*) – Name of the dataset to be added.
- **dataset_specs** (*dict*) – Specifications of the dataset.

Returns None

pysemantic.project.**add_project**(*project_name*, *specfile*)
Add a project to the global configuration file.

Parameters

- **project_name** (*str*) – Name of the project
- **specfile** (*str*) – path to the data dictionary used by the project.

Returns None

pysemantic.project.**get_datasets**(*project_name=None*)
Get names of all datasets registered under the project *project_name*.

Parameters **project_name** (*str*) – name of the projects to list the datasets from. If *None* (default), datasets under all projects are returned.

Returns List of datasets listed under *project_name*, or if *project_name* is *None*, returns dictionary such that {project_name: [list of projects]}

Return type dict or list

Example

```
>>> get_datasets('skynet')
['sarah_connor', 'john_connor', 'kyle_reese']
>>> get_datasets()
{'skynet': ['sarah_connor', 'john_connor', 'kyle_reese'],
 'south park': ['stan', 'kyle', 'cartman', 'kenny']}
```

pysemantic.project.**get_default_specfile**(*project_name*)

Returns the specifications file used by the given project. The configuration file is searched for first in the current directory and then in the home directory.

Parameters **project_name** (*str*) – Name of the project for which to get the spcfile.

Returns Path to the data dictionary of the project.

Return type str

Example

```
>>> get_default_specfile('skynet')
'/home/username/projects/skynet/schema.yaml'
```

pysemantic.project.**get_projects**()

Get the list of projects currently registered with pysemantic as a list.

Returns List of tuples, such that each tuple is (project_name, location_of_specfile)

Return type list

Example

```
>>> get_projects()
['skynet', 'south park']
```

pysemantic.project.**get_schema_specs**(*project_name*, *dataset_name=None*)

Get the specifications of a dataset as specified in the schema.

Parameters

- **project_name** (*str*) – Name of project
- **dataset_name** (*str*) – name of the dataset for which to get the schema. If None (default), schema for all datasets is returned.

Returns schema for dataset

Return type dict

Example

```
>>> get_schema_specs('skynet')
{'sarah connor': {'path': '/path/to/sarah_connor.csv',
                  'delimiter': ','},
 'kyle reese': {'path': '/path/to/kyle_reese.tsv',
                'delimiter':, '     '}
 'john connor': {'path': '/path/to/john_connor.txt',
```

```
                    'delimiter':, ' '}
    }
```

pysemantic.project.**locate_config_file**()
Locates the configuration file used by semantic.

Returns Path of the pysemantic config file.

Return type str

Example

```
>>> locate_config_file()
'/home/username/pysemantic.conf'
```

pysemantic.project.**remove_dataset**(*project_name*, *dataset_name*)
Removes a dataset from a project.

Parameters

- **project_name** (*str*) – Name of the project
- **dataset_name** (*str*) – Name of the dataset to remove

Returns None

pysemantic.project.**remove_project**(*project_name*)
Remove a project from the global configuration file.

Parameters **project_name** (*str*) – Name of the project to remove.

Returns True if the project existed

Return type bool

Example

```
>>> view_projects()
Project skynet with specfile at /path/to/skynet.yaml
Project south park with specfile at /path/to/south_park.yaml
>>> remove_project('skynet')
>>> view_projects()
Project south park with specfile at /path/to/south_park.yaml
```

pysemantic.project.**set_schema_fpath**(*project_name*, *schema_fpath*)
Set the schema path for a given project.

Parameters

- **project_name** (*str*) – Name of the project
- **schema_fpath** (*str*) – path to the yaml file to be used as the schema for the project.

Returns True, if setting the schema path was successful.

Example

```
>>> set_schema_fpath('skynet', '/path/to/new/schema.yaml')
True
```

pysemantic.project.**set_schema_specs**(*project_name*, *dataset_name*, ***kwargs*)
Set the schema specifications for a dataset.

Parameters

- **project_name** (*str*) – Name of the project containing the dataset.

- **dataset_name** (*str*) – Name of the dataset of which the schema is being set.
- **kwargs** – Schema fields that are dumped into the schema files.

Returns None

Example

```
>>> set_schema_specs('skynet', 'kyle reese',
                     path='/path/to/new/file.csv', delimiter=new_delimiter)
```

pysemantic.project.**view_projects**()
View a list of all projects currently registered with pysemantic.

Example

```
>>> view_projects()
Project skynet with specfile at /path/to/skynet.yaml
Project south park with specfile at /path/to/south_park.yaml
```

3.1.8 pysemantic.utils module

Misecellaneous bells and whistles.

class pysemantic.utils.**TypeEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
Bases: json.encoder.JSONEncoder

default(*obj*)

pysemantic.utils.**colnames**(*filename*, ***kwargs*)
Read the column names of a delimited file, without actually reading the whole file. This is simply a wrapper around *pandas.read_csv*, which reads only one row and returns the column names.

Parameters

- **filename** (*str*) – Path to the file to be read
- **kwargs** – Arguments to be passed to the *pandas.read_csv*

Return type list

Example

Suppose we want to see the column names of the Fisher iris dataset.

```
>>> colnames("/path/to/iris.csv")
['Sepal Length', 'Petal Length', 'Sepal Width', 'Petal Width', 'Species']
```

pysemantic.utils.**generate_questionnaire**(*filepath*)
Generate a questionnaire for data at *filepath*.

This questionnaire will be presented to the client, which helps us automatically generate the schema.

Parameters **filepath** (*str*) – Path to the file that needs to be ingested.

Returns A dictionary of questions and their possible answers. The format

of the dictionary is such that every key is a question to be put to the client, and its value is a list of possible answers. The first item in the list is the default value. :rtype: dict

pysemantic.utils.**get_md5_checksum**(*filepath*)
Get the md5 checksum of a file.

Parameters **filepath** (*Str*) – Path to the file of which to calculate the md5 checksum.

Returns MD5 checksum of the file.

Return type Str

Example

```
>>> get_md5_checksum('pysemantic/tests/testdata/iris.csv')
'9b3ecf3031979169c0ecc5e03cfe20a6'
```

3.1.9 pysemantic.validator module

Traited Data validator for *pandas.DataFrame* objects.

class pysemantic.validator.**DataFrameValidator**
Bases: traits.has_traits.HasTraits

A validator class for *pandas.DataFrame* objects.

clean()
Return the converted dataframe after enforcing all rules.

rename_columns()
Rename columns in dataframe as per the schema.

class pysemantic.validator.**SchemaValidator**(***kwargs*)
Bases: traits.has_traits.HasTraits

A validator class for schema in the data dictionary.

classmethod from_dict(*specification*)
Get a validator from a schema dictionary.

Parameters **specification** – Dictionary containing schema specifications.

classmethod from_specfile(*specfile*, *name*, ***kwargs*)
Get a validator from a schema file.

Parameters

- **specfile** – Path to the schema file.
- **name** – Name of the project to create the validator for.

get_parser_args()
Return parser args as required by pandas parsers.

set_parser_args(*specs*, *write_to_file=False*)
Magic method required by Property traits.

to_dict()
Return parser args as required by pandas parsers.

class pysemantic.validator.**SeriesValidator**
Bases: traits.has_traits.HasTraits

A validator class for *pandas.Series* objects.

apply_minmax_rules()
Restrict the series to the minimum and maximum from the schema.

apply_regex()
Apply a regex filter on strings in the series.

apply_uniques()
Remove all values not included in the *uniques*.

clean()
Return the converted dataframe after enforcing all rules.

do_drop_duplicates()
Drop duplicates from the series if required.

do_drop_na()
Drop NAs from the series if required.

do_postprocessing()

drop_excluded()
Remove all values specified in *exclude_values*.

3.1.10 Module contents

class pysemantic.**Project**(*project_name=None*, *parser=None*, *schema=None*)
Bases: object

The Project class, the entry point for most things in this module.

datasets
"List the datasets registered under the parent project.

Example

```
>>> project = Project('skynet')
>>> project.datasets
['sarah connor', 'john connor', 'kyle reese']
```

export_dataset(*dataset_name*, *dataframe=None*, *outpath=None*)
Export a dataset to an exporter defined in the schema. If nothing is specified in the schema, simply export to a CSV file such named <dataset_name>.csv

Parameters

- **dataset_name** (*Str*) – Name of the dataset to exporter.
- **dataframe** – Pandas dataframe to export. If None (default), this dataframe is loaded using the *load_dataset* method.

get_dataset_specs(*dataset_name*)
Returns the specifications for the specified dataset in the project.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns Parser arguments required to import the dataset in pandas.

Return type dict

get_project_specs()
Returns a dictionary containing the schema for all datasets listed under this project.

Returns Parser arguments for all datasets listed under the project.

Return type dict

load_dataset(*dataset_name*)
Load and return a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns A pandas DataFrame containing the dataset.

Return type pandas.DataFrame

Example

```
>>> demo_project = Project('pysemantic_demo')
>>> iris = demo_project.load_dataset('iris')
>>> type(iris)
pandas.core.DataFrame
```

load_datasets()
Load and return all datasets.

Returns dictionary like {dataset_name: dataframe}

Return type dict

reload_data_dict()
Reload the data dictionary and re-populate the schema.

set_dataset_specs(*dataset_name*, *specs*, *write_to_file=False*)
Sets the specifications to the dataset. Using this is not recommended. All specifications for datasets should be handled through the data dictionary.

Parameters

- **dataset_name** (*str*) – Name of the dataset for which specifications need to be modified.
- **specs** (*dict*) – A dictionary containing the new specifications for the dataset.
- **write_to_file** (*bool*) – If true, the data dictionary will be updated to the new specifications. If False (the default), the new specifications are used for the respective dataset only for the lifetime of the *Project* object.

Returns None

update_dataset(*dataset_name*, *dataframe*, *path=None*, ***kwargs*)
This is tricky.

view_dataset_specs(*dataset_name*)
Pretty print the specifications for a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

pysemantic.**test**()
Interactive loader for tests.

3.2 pysemantic

3.2.1 pysemantic package

Submodules

pysemantic.cli module

semantic

Usage: semantic list [–project=<PROJECT_NAME>] semantic add PROJECT_NAME PROJECT_SPECFILE semantic remove PROJECT_NAME [–dataset=<dname>] semantic set-schema PROJECT_NAME SCHEMA_FPATH semantic set-specs PROJECT_NAME –dataset=<dname> [–path=<pth>] [–dlm=<sep>]

semantic add-dataset DATASET_NAME –project=<pname> –path=<pth> –dlm=<sep> semantic export PROJECT_NAME [–dataset=<dname>] OUTPATH

Options: -h –help Show this screen -d –dataset=<dname> Name of the dataset to modify –path=<pth> Path to a dataset –dlm=<sep> Declare the delimiter for a dataset -p –project=<pname> Name of the project to modify -v –version Print the version of PySemantic

pysemantic.cli.**cli**(*arguments*)

cli - The main CLI argument parser.

Parameters **arguments** (*dict*) – command line arguments, as parsed by docopt

Returns None

pysemantic.cli.**main**()

pysemantic.custom_traits module

Customized traits for advanced validation.

class pysemantic.custom_traits.**AbsFile**(*value=''*, *filter=None*, *auto_set=False*, *entries=0*, *exists=False*, ***metadata*)

Bases: traits.trait_types.File

A File trait whose value must be an absolute path, to an existing file.

validate(*obj*, *name*, *value*)

class pysemantic.custom_traits.**DTypeTraitDictObject**(*trait*, *object*, *name*, *value*)

Bases: traits.trait_handlers.TraitDictObject

Subclassed from the parent to aid the validation of DTypesDicts.

class pysemantic.custom_traits.**DTypesDict**(*key_trait=None*, *value_trait=None*, *value=None*, *items=True*, ***metadata*)

Bases: traits.trait_types.Dict

A trait whose keys are strings, and values are Type traits. Ideally this is the kind of dictionary that is passed as the *dtypes* argument in *pandas.read_table*.

validate(*obj*, *name*, *value*)

Subclassed from the parent to return a *DTypeTraitDictObject* instead of *traits.trait_handlers.TraitDictObhject*.

class pysemantic.custom_traits.**NaturalNumber**(*default_value=<traits.trait_handlers.NoDefaultSpecified object>*, ***metadata*)

Bases: traits.trait_types.BaseInt

An integer trait whose value is a natural number.

default_value = 1

error(*obj*, *name*, *value*)

validate(*obj*, *name*, *value*)

class pysemantic.custom_traits.**ValidTraitList**(*trait=None*, *value=None*, *minlen=0*, *maxlen=9223372036854775807*, *items=True*, ***metadata*)

Bases: traits.trait_types.List

A List trait whose every element should be valid trait.

validate(*obj*, *name*, *value*)

pysemantic.errors module

Errors.

exception pysemantic.errors.**MissingConfigError**
Bases: exceptions.Exception

Error raised when the pysemantic configuration file is not found.

exception pysemantic.errors.**MissingProject**
Bases: exceptions.Exception

Error raised when project is not found.

pysemantic.exporters module

Exporters from PySemantic to databases or other data sinks.

class pysemantic.exporters.**AbstractExporter**
Bases: object

Abstract exporter for dataframes that have been cleaned.

get(***kwargs*)

set(***kwargs*)

class pysemantic.exporters.**AerospikeExporter**(*config*, *dataframe*)
Bases: *pysemantic.exporters.AbstractExporter*

Example class for exporting to an aerospike database.

run()

set(*key_tuple*, *bins*)

pysemantic.loggers module

Loggers

pysemantic.loggers.**setup_logging**(*project_name*)

pysemantic.project module

The Project class.

class pysemantic.project.**Project**(*project_name=None*, *parser=None*, *schema=None*)
Bases: object

The Project class, the entry point for most things in this module.

datasets
"List the datasets registered under the parent project.

Example

```
>>> project = Project('skynet')
>>> project.datasets
['sarah connor', 'john connor', 'kyle reese']
```

export_dataset(*dataset_name*, *dataframe=None*, *outpath=None*)
Export a dataset to an exporter defined in the schema. If nothing is specified in the schema, simply export to a CSV file such named <dataset_name>.csv

Parameters

- **dataset_name** (*Str*) – Name of the dataset to exporter.
- **dataframe** – Pandas dataframe to export. If None (default), this dataframe is loaded using the *load_dataset* method.

get_dataset_specs(*dataset_name*)
Returns the specifications for the specified dataset in the project.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns Parser arguments required to import the dataset in pandas.

Return type dict

get_project_specs()
Returns a dictionary containing the schema for all datasets listed under this project.

Returns Parser arguments for all datasets listed under the project.

Return type dict

load_dataset(*dataset_name*)
Load and return a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns A pandas DataFrame containing the dataset.

Return type pandas.DataFrame

Example

```
>>> demo_project = Project('pysemantic_demo')
>>> iris = demo_project.load_dataset('iris')
>>> type(iris)
pandas.core.DataFrame
```

load_datasets()
Load and return all datasets.

Returns dictionary like {dataset_name: dataframe}

Return type dict

reload_data_dict()
Reload the data dictionary and re-populate the schema.

set_dataset_specs(*dataset_name*, *specs*, *write_to_file=False*)
Sets the specifications to the dataset. Using this is not recommended. All specifications for datasets should be handled through the data dictionary.

Parameters

- **dataset_name** (*str*) – Name of the dataset for which specifications need to be modified.
- **specs** (*dict*) – A dictionary containing the new specifications for the dataset.
- **write_to_file** (*bool*) – If true, the data dictionary will be updated to the new specifications. If False (the default), the new specifications are used for the respective dataset only for the lifetime of the *Project* object.

Returns None

update_dataset (*dataset_name*, *dataframe*, *path=None*, ***kwargs*)
This is tricky.

view_dataset_specs (*dataset_name*)
Pretty print the specifications for a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

pysemantic.project.**add_dataset** (*project_name*, *dataset_name*, *dataset_specs*)
Add a dataset to a project.

Parameters

- **project_name** (*str*) – Name of the project to which the dataset is to be added.
- **dataset_name** (*str*) – Name of the dataset to be added.
- **dataset_specs** (*dict*) – Specifications of the dataset.

Returns None

pysemantic.project.**add_project** (*project_name*, *specfile*)
Add a project to the global configuration file.

Parameters

- **project_name** (*str*) – Name of the project
- **specfile** (*str*) – path to the data dictionary used by the project.

Returns None

pysemantic.project.**get_datasets** (*project_name=None*)
Get names of all datasets registered under the project *project_name*.

Parameters **project_name** (*str*) – name of the projects to list the datasets from. If *None* (default), datasets under all projects are returned.

Returns List of datasets listed under *project_name*, or if *project_name* is *None*, returns dictionary such that {project_name: [list of projects]}

Return type dict or list

Example

```
>>> get_datasets('skynet')
['sarah_connor', 'john_connor', 'kyle_reese']
>>> get_datasets()
{'skynet': ['sarah_connor', 'john_connor', 'kyle_reese'],
 'south park': ['stan', 'kyle', 'cartman', 'kenny']}
```

pysemantic.project.**get_default_specfile** (*project_name*)
Returns the specifications file used by the given project. The configuration file is searched for first in the current directory and then in the home directory.

Parameters **project_name** (*str*) – Name of the project for which to get the spcfile.

Returns Path to the data dictionary of the project.

Return type str

Example

```
>>> get_default_specfile('skynet')
'/home/username/projects/skynet/schema.yaml'
```

pysemantic.project.**get_projects**()
Get the list of projects currently registered with pysemantic as a list.

Returns List of tuples, such that each tuple is (project_name, location_of_specfile)

Return type list

Example

```
>>> get_projects()
['skynet', 'south park']
```

pysemantic.project.**get_schema_specs**(*project_name*, *dataset_name=None*)
Get the specifications of a dataset as specified in the schema.

Parameters

- **project_name** (*str*) – Name of project
- **dataset_name** (*str*) – name of the dataset for which to get the schema. If None (default), schema for all datasets is returned.

Returns schema for dataset

Return type dict

Example

```
>>> get_schema_specs('skynet')
{'sarah connor': {'path': '/path/to/sarah_connor.csv',
                  'delimiter': ','},
 'kyle reese': {'path': '/path/to/kyle_reese.tsv',
                'delimiter':, '     '}
 'john connor': {'path': '/path/to/john_connor.txt',
                'delimiter':, ' '}
 }
```

pysemantic.project.**locate_config_file**()
Locates the configuration file used by semantic.

Returns Path of the pysemantic config file.

Return type str

Example

```
>>> locate_config_file()
'/home/username/pysemantic.conf'
```

pysemantic.project.**remove_dataset**(*project_name*, *dataset_name*)
Removes a dataset from a project.

Parameters

- **project_name** (*str*) – Name of the project
- **dataset_name** (*str*) – Name of the dataset to remove

Returns None

pysemantic.project.**remove_project**(*project_name*)
Remove a project from the global configuration file.

Parameters **project_name** (*str*) – Name of the project to remove.

Returns True if the project existed

Return type bool

Example

```
>>> view_projects()
Project skynet with specfile at /path/to/skynet.yaml
Project south park with specfile at /path/to/south_park.yaml
>>> remove_project('skynet')
>>> view_projects()
Project south park with specfile at /path/to/south_park.yaml
```

pysemantic.project.**set_schema_fpath**(*project_name*, *schema_fpath*)

Set the schema path for a given project.

Parameters

- **project_name** (*str*) – Name of the project
- **schema_fpath** (*str*) – path to the yaml file to be used as the schema for the project.

Returns True, if setting the schema path was successful.

Example

```
>>> set_schema_fpath('skynet', '/path/to/new/schema.yaml')
True
```

pysemantic.project.**set_schema_specs**(*project_name*, *dataset_name*, ***kwargs*)

Set the schema specifications for a dataset.

Parameters

- **project_name** (*str*) – Name of the project containing the dataset.
- **dataset_name** (*str*) – Name of the dataset of which the schema is being set.
- **kwargs** – Schema fields that are dumped into the schema files.

Returns None

Example

```
>>> set_schema_specs('skynet', 'kyle reese',
                     path='/path/to/new/file.csv', delimiter=new_delimiter)
```

pysemantic.project.**view_projects**()

View a list of all projects currently registered with pysemantic.

Example

```
>>> view_projects()
Project skynet with specfile at /path/to/skynet.yaml
Project south park with specfile at /path/to/south_park.yaml
```

pysemantic.utils module

Misecellaneous bells and whistles.

class pysemantic.utils.**TypeEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Bases: json.encoder.JSONEncoder

default(*obj*)

pysemantic.utils.**colnames**(*filename*, ***kwargs*)

Read the column names of a delimited file, without actually reading the whole file. This is simply a wrapper around *pandas.read_csv*, which reads only one row and returns the column names.

Parameters

- **filename** (*str*) – Path to the file to be read
- **kwargs** – Arguments to be passed to the *pandas.read_csv*

Return type list

Example

Suppose we want to see the column names of the Fisher iris dataset.

```
>>> colnames("/path/to/iris.csv")
['Sepal Length', 'Petal Length', 'Sepal Width', 'Petal Width', 'Species']
```

pysemantic.utils.**generate_questionnaire**(*filepath*)

Generate a questionnaire for data at *filepath*.

This questionnaire will be presented to the client, which helps us automatically generate the schema.

Parameters **filepath** (*str*) – Path to the file that needs to be ingested.

Returns A dictionary of questions and their possible answers. The format

of the dictionary is such that every key is a question to be put to the client, and its value is a list of possible answers. The first item in the list is the default value. :rtype: dict

pysemantic.utils.**get_md5_checksum**(*filepath*)

Get the md5 checksum of a file.

Parameters **filepath** (*Str*) – Path to the file of which to calculate the md5 checksum.

Returns MD5 checksum of the file.

Return type Str

Example

```
>>> get_md5_checksum('pysemantic/tests/testdata/iris.csv')
'9b3ecf3031979169c0ecc5e03cfe20a6'
```

pysemantic.validator module

Traited Data validator for *pandas.DataFrame* objects.

class pysemantic.validator.**DataFrameValidator**

Bases: traits.has_traits.HasTraits

A validator class for *pandas.DataFrame* objects.

clean()

Return the converted dataframe after enforcing all rules.

rename_columns()
Rename columns in dataframe as per the schema.

class pysemantic.validator.**SchemaValidator**(***kwargs*)
Bases: traits.has_traits.HasTraits

A validator class for schema in the data dictionary.

classmethod from_dict(*specification*)
Get a validator from a schema dictionary.

Parameters **specification** – Dictionary containing schema specifications.

classmethod from_specfile(*specfile*, *name*, ***kwargs*)
Get a validator from a schema file.

Parameters

- **specfile** – Path to the schema file.
- **name** – Name of the project to create the validator for.

get_parser_args()
Return parser args as required by pandas parsers.

set_parser_args(*specs*, *write_to_file=False*)
Magic method required by Property traits.

to_dict()
Return parser args as required by pandas parsers.

class pysemantic.validator.**SeriesValidator**
Bases: traits.has_traits.HasTraits

A validator class for *pandas.Series* objects.

apply_minmax_rules()
Restrict the series to the minimum and maximum from the schema.

apply_regex()
Apply a regex filter on strings in the series.

apply_uniques()
Remove all values not included in the *uniques*.

clean()
Return the converted dataframe after enforcing all rules.

do_drop_duplicates()
Drop duplicates from the series if required.

do_drop_na()
Drop NAs from the series if required.

do_postprocessing()

drop_excluded()
Remove all values specified in *exclude_values*.

Module contents

class pysemantic.**Project**(*project_name=None*, *parser=None*, *schema=None*)
Bases: object

The Project class, the entry point for most things in this module.

datasets
"List the datasets registered under the parent project.

Example

```
>>> project = Project('skynet')
>>> project.datasets
['sarah connor', 'john connor', 'kyle reese']
```

export_dataset(*dataset_name*, *dataframe=None*, *outpath=None*)
Export a dataset to an exporter defined in the schema. If nothing is specified in the schema, simply export to a CSV file such named <dataset_name>.csv

Parameters

- **dataset_name** (*Str*) – Name of the dataset to exporter.
- **dataframe** – Pandas dataframe to export. If None (default), this dataframe is loaded using the *load_dataset* method.

get_dataset_specs(*dataset_name*)
Returns the specifications for the specified dataset in the project.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns Parser arguments required to import the dataset in pandas.

Return type dict

get_project_specs()
Returns a dictionary containing the schema for all datasets listed under this project.

Returns Parser arguments for all datasets listed under the project.

Return type dict

load_dataset(*dataset_name*)
Load and return a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

Returns A pandas DataFrame containing the dataset.

Return type pandas.DataFrame

Example

```
>>> demo_project = Project('pysemantic_demo')
>>> iris = demo_project.load_dataset('iris')
>>> type(iris)
pandas.core.DataFrame
```

load_datasets()
Load and return all datasets.

Returns dictionary like {dataset_name: dataframe}

Return type dict

reload_data_dict()
Reload the data dictionary and re-populate the schema.

set_dataset_specs(*dataset_name*, *specs*, *write_to_file=False*)
Sets the specifications to the dataset. Using this is not recommended. All specifications for datasets should be handled through the data dictionary.

Parameters

- **dataset_name** (*str*) – Name of the dataset for which specifications need to be modified.
- **specs** (*dict*) – A dictionary containing the new specifications for the dataset.
- **write_to_file** (*bool*) – If true, the data dictionary will be updated to the new specifications. If False (the default), the new specifications are used for the respective dataset only for the lifetime of the *Project* object.

Returns None

update_dataset(*dataset_name*, *dataframe*, *path=None*, ***kwargs*)
This is tricky.

view_dataset_specs(*dataset_name*)
Pretty print the specifications for a dataset.

Parameters **dataset_name** (*str*) – Name of the dataset

pysemantic.**test**()
Interactive loader for tests.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

p

A

C

D

E

F

G

L

M

N

P

R

S

T

U

V