Tabs Documentation

Release 0.6.4

Ole Henrik Skogstrøm

Nov 03, 2017

Table of Contents

Tabs is a small framework for defining and loading tables in a consistent way. The goal is to make data science projects more maintainable by improving code readability.

Tabs comes with support for caching processed tables based on the current configuration resulting in shorter loading of tables that have already been compiled once.

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

1.1 Basic concepts

Tabs consists of two main classes.

- Tabs
- Table

1.1.1 Table

Table is an abstract class used to define new tables. This ensures that all tables has a minimum of shared functionality, like fetching a table or describing it.

class `tabs.`**`Table`**(**args*, ***kwargs*)
MetaClass for defining tables.

Attention! The following methods are required when defining a class the inherits from Table

`source`(*self*)
Should return the table. For example pd.read_csv() **(required, method)**

`output`(*self*)
Should return the output path for where the finished table should be stored. For example a cache directory. **(required, method)**

`post_processors`(*self*)
a list of post processor functions of methods. **(required, method)**

1.1.1.0.1 Example

Defining a table:

```
class UserDataTable(Table):
    def source(self):
        return pd.read_csv('/path/to/file')

    def output(self):
        return "/path/to/output"

    def post_processors(self):
        return [
            my_custom_function(),
            my_second_custom_function(),
        ]
```

1.1.2 Tabs

Tabs is the class used to load all tables defined in a package. This is the class used for loading tables and gaining an overview of all tables defined in a package.

class tabs.**Tabs**(*package_path=None*, *custom_table_classes=None*)

Class for loading a list of all defined tables, similar to tabs in a browser.

Parameters

- **package_path** (*str*) – Path to package containing defined tables
- **custom_table_classes** (*list(class)*) – A list of custom Table metaclasses that should also be recognised and added to the tabs list.

1.1.2.0.1 Example

Using tabs for listing tables:

```
from tabs import Tabs
package_path = os.path.dirname(os.path.realpath(__file__))
tabs = Tabs(package_path)
tabs.table_list()

> Avaiable tables:
> Persondata
> OtherData
```

Fetching a defined table:

```
person_data = tabs('Persondata').fetch()
```

1.2 Usage - Tabs explained

Usage of tabs is best shown through an example. In the following example the project has this folder structure:

```
csv_files/
  |- example_file_one.csv
  |- example_file_one.csv
output/
table_definition.py
table_usage.py
```

1.2.1 Table

Defining a table:

```
# in /table_definition.py

import os
from datetime import datetime
from tabs import Table
from dateutil.relativedelta import relativedelta
import pandas as pd
import numpy as np

def drop_age_column(table):
    """Drops age from original dataframe because of wrong age """
    table.drop('age', 1, inplace=True)
    return table

def calculate_new_age(table):
    """Calculates new age and adds it to the dataframe"""
    date_now = datetime.now()
    def get_age(birthday):
        if birthday:
            return relativedelta(date_now, birthday).years
    table['age'] = table.apply(lambda birthday: get_age)
    return table

class TestTableOne(Table):
    """Table containing names, birthday and age of participants"""
    def source(self):
        source_file = os.path.join(os.path.dirname(os.path.realpath(__file__)),
                                   'csv_files',
                                   'test_table_one.csv')
        dtype = {
            'first': np.str,
            'last': np.str,
            'age': np.int
        }

        converters = {
            'birthday': pd.to_datetime,
        }

        return pd.read_csv(source_file, dtype=dtype, converters=converters)

    def output(self):
        output_path = os.path.join(os.path.dirname(os.path.realpath(__file__)),
                                   'output',
                                   self.get_cached_filename('test_table_one', 'pkl')
```

```
                                    )
        return output_path

    def post_processors(self):
        return [
            drop_age_column,
            calculate_new_age
        ]
```

Here you should first pay attention to the class `TestTableOne`. This inherits from the abstract class `Table` that requires `source`, `output` and `post_processors` to be defined.

`source` is used to define how the table is loaded before any post processors are applied.

`output` specifies where the table is stored and if it utilizes the `get_cached_filename` method that applies a hash id based on the content of `source`, *output* and *post_processors*. This ensures that if the table is modified either through source, output or post processors, the table is regenerated.

`post_processors` is an array of functions that takes the complete table as an source and returns a modified table. This is where you instruct what changes you apply to your table and in what order.

1.2.2 Tabs

The `Tabs` class can be used to load tables and getting an overview of which tables are defined and how they are processed:

```
# in /table_usage.py
from tabs import Tabs
package_path = os.path.dirname(os.path.realpath(__file__))
tabs = Tabs(package_path)
test_table_one = tabs('TestTableOne').fetch()

len(test_table_one) # >>>> 100
list(test_table_one) # >>>> ['first', 'last', 'birthday', 'age']
test_table_one.head() # test_table_one is a normal pandas table

# This will print a list of all defined tables and their post porcessors.
tabs.describe_all(full=True)
```

1.2.3 Table and Tabs - Utility methods

1.2.3.1 describe

Is either used directly on defined tables (i.e. TestTableOne) or through Tables and will print out a description of the table based on the *__doc__* defined in the class. If *full=True* is provided the post processors and their description will also be included.

Example with TestTableOne: `TestTableOne.describe(full=True)`

Example through Tabs: `Tabs(package_path)('TestTableOne').describe(full=True)`

1.2.3.2 describe_all

Does the same as *describe* but for all defined tables. Only exists on Tabs.

1.2.3.3 fetch

Is either used directly on defined tables (i.e. TestTableOne) or through Tabs and is used to fetch the pandas table from the a defined table.

Example with TestTableOne: `TestTableOne().fetch()`

Example through Tabs: `Tabs(package_path)('TestTableOne').fetch()`

1.2.3.4 get_cached_filename

Is used inside the *output* method to add a hash id after the output filename.

`self.get_cached_filename('test_table_one', 'pkl')` will return something similar to `test_table_one_1341423423fds23.pkl` based on what configurations you have applied.

Exmaple:

```
def output(self):
    output_path = os.path.join(os.path.dirname(os.path.realpath(__file__)),
                               'output',
                               self.get_cached_filename('test_table_one', 'pkl')
                              )
    return output_path
```

1.3 tabs

1.3.1 tabs package

1.3.1.1 Submodules

1.3.1.2 tabs.tables module

Table base classes for defning new tables

class `tabs.tables.`**`BaseTableABC`**(**args*, ***kwargs*)

Bases: `object`

Abstract Base class for minimum table import

classmethod `dep`()

dep is an alias of dependencies

classmethod `dependencies`()

Returns a list of all dependent tables, in the order they are defined.

Add new dependencies for source and every post proecssor like this:

```
source.dependencies = [PersonalData]
some_post_processor.dependencies = [SomeOtherTable, AnotherTable]
```

some_post_processor.dependencies needs to be placed after *some_post_processor* is defined.

classmethod `describe`(*full=False*)

Prints a description of the table based on the provided documentation and post processors.

Parameters **full** (*bool*) – Include post processors in the printed description.

classmethod describe_processors()
List all postprocessors and their description

fetch(*rebuild=False*, *cache=True*)
Method for fetching data

get_cached_filename(*filename*, *extention*, *settings_list=None*)
Creates a filename with md5 cache string based on settings list

Parameters

- **filename** (*str*) – the filename without extention
- **extention** (*str*) – the file extention without dot. (i.e. 'pkl')
- **settings_list** (*dict | list*) – the settings list as list (optional) NB! The dictionaries have to be sorted or hash id will change arbitrarely.

get_hash()
Retruns a hash based on the the current table code and kwargs. Also changes based on dependent tables.

get_settings_list()
The settings list used for building the cache id.

output()
Path to the processed table (output path)

post_processors()
A list of functions to be applied for post processing

source()
Path to the original raw data

class tabs.tables.**Table**(**args*, ***kwargs*)
Bases: *tabs.tables.BaseTableABC*

MetaClass for defining tables.

Attention! The following methods are required when defining a class the inherits from Table

source(*self*)
Should return the table. For example pd.read_csv() **(required, method)**

output(*self*)
Should return the output path for where the finished table should be stored. For example a cache directory. **(required, method)**

post_processors(*self*)
a list of post processor functions of methods. **(required, method)**

1.3.1.2.1 Example

Defining a table:

```
class UserDataTable(Table):
    def source(self):
        return pd.read_csv('/path/to/file')

    def output(self):
        return "/path/to/output"
```

```
    def post_processors(self):
        return [
            my_custom_function(),
            my_second_custom_function(),
        ]
```

fetch(*rebuild=False*, *cache=True*)
Fetches the table and applies all post processors. :param rebuild: Rebuild the table and ignore cache. Default: False :type rebuild: bool :param cache: Cache the finished table for faster future loading.

Default: True

output()
Path to the processed table (output path)

post_processors()
A list of functions to be applied for post processing

read_cache()
Defines how to read table from cache. Should be overwritten if to cache is overwritten

source()
Path to the original raw data

to_cache(*table*)
Defines the default cache method. Can be overwritten if needed

tabs.tables.**describe**(*cls*, *full=False*)
Prints a description of the table based on the provided documentation and post processors

tabs.tables.**post_process**(*table*, *post_processors*)
Applies the list of post processing methods if any

1.3.1.3 tabs.tabs module

Tables module

class tabs.tabs.**Tabs**(*package_path=None*, *custom_table_classes=None*)
Bases: object

Class for loading a list of all defined tables, similar to tabs in a browser.

Parameters

- **package_path** (*str*) – Path to package containing defined tables
- **custom_table_classes** (*list(class)*) – A list of custom Table metaclasses that should also be recognised and added to the tabs list.

1.3.1.3.1 Example

Using tabs for listing tables:

```
from tabs import Tabs
package_path = os.path.dirname(os.path.realpath(__file__))
tabs = Tabs(package_path)
tabs.table_list()

> Avaiable tables:
> Persondata
> OtherData
```

Fetching a defined table:

```
person_data = tabs('Persondata').fetch()
```

describe_all (*full=False*)
Prints description information about all tables registered :param full: Also prints description of post processors. :type full: bool

find_tabs (*custom_table_classes=None*)
Finds all classes that are subcalss of Table and loads them into a dictionary named tables.

get (*table_name*)
Load table class by name, class not yet initialized

load (*table_name*, ***kwargs*)
Get table object by name, initialized and ready. Same as using __call__

table_list ()
Display the table names

tabs.tabs.**get_all_classes** (*module_name*)
Load all non-abstract classes from package

tabs.tabs.**get_all_modules** (*package_path*)
Load all modules in a package

1.3.1.4 Module contents

Tabs

Python Module Index

t

Index