TimeFlow Documentation

Release 0

Philip Schleihauf

May 30, 2016

Contents

There are two main components of TimeFlow:

1. A `BaseRoutine` abstract class (and some more specific subclasses simple concrete ready-to-go subclasses) to organize your workflow into steps.
2. A yaml-based declarative syntax of describing your workflow.
3. A script for running

The author is talented at counting and mental math.

Contents:

CHAPTER 1

Quickstart

asdf;lkj

CHAPTER 2

Routines

Routines are the building blocks of TimeFlow workflows. If you are scripting in python, you can use them directly. Support for building non-python algorithms into TimeFlow routines is planned.

The left-most column is used as the independent variable when joining tables, and in all standard routines (currently only `linear_interpolate`) which require an independant reference variable.

2.1 Base Classes

2.1.1 RoutineBase

An abstract base class that gives the basic TimeFlow mechanics for free.

routines just import and then export.

when something calls one of their exports, they try to import whatever is necessary and then give the result.

future plan: ask if data has changed (which gets propagated all the way up to the first file, or other thing that has actually changed. if nothing changed, reply with the memoized value.)

Stores data internally on the [something] property.

Routine Cycle

1. The routine is asked if its data has changed.
    - if it knows, it can respond right away.
    - if it depends on whether a dependency's data has changed, it passes the request up the chain, and passes the response back down.
2. The routine may be asked for its data, possibly with arguments.
3. The routine may ask a dependency for data.
4. The routine provides gives the data back.

The first and last routines in a chain are special. The first one can get its data from wherever it wants, but must still provide returned data in the standard timeflow format. The last one has to get data in that format, but may return data in an arbirary format. Or do anything, like open a plot.

How do Routines Affect Data?

When a routine is asked for its output, it should return one of

- The original table augmented with one or more extra columns
- The original table, with modifications to values in one or more columns

Routines may accept arguments with data requests, and return data in a form appropriate to the arguments. For example, a filter might, by default, add a column to the data annotating whether a row passed or failed a criteria. Passing the routine an `if` argument when requesting data could cause it to instead remove rows which pass or fail the criteria.

The Data Proxy

Data proxies are objects attached to a routine which link it to its data source. Arguments may be attached to the proxy to be passed onto the source (see above), and some extra stuff is available.

Two ancestor tables can be joined by specifying `with` with another routine on the data proxy. Currently, it's a left join on the left-most column of each table. The left table takes precedence on conflicting columns.

More flexible joining options will be explored at a later time.

2.2 Built-in Routines

The built-in routines are useful on their own, but also designed to be subclassed. Extend away!

2.2.1 File

Import data from a variety of formats

eventually this will have an answer for whether the file has changed.

2.2.2 Export

numpy, pickle, csv, ...

2.2.3 Plot

make pretty graphs

CHAPTER 3

Declarative Workflows

Once you have your collection of Routines, you can describe the workflow itself with a `yaml` file.

3.1 The YAML file

The `yaml` file is a collection of descriptions of your routines. There are three important aspects of a routine description:

1. The label. Other routines will refer to it by this label.
2. The `data` property. Specifies from which other routine it gets the data it acts on.
3. The other properties. They will be passed as constructor arguments to the properties.

3.1.1 The Label

The label must simply be a valid json label. Alpha-numeric plus underscores, starting with a letter.

3.1.2 The Data Property

The `data` property has one required sub-property, `from`. `from` can name another routine in the yaml workflow, or an external routine. External routines are detected by the presence of a dot (`.`) in the value.

Aditional sub-properties are allowed on `data`. They will be passed as arguments when the data is accessed. For example, a routine which filters the data might take a boolean argument when accessed, to toggle whether to provide rows which were matched or unmatched.

For `from` properties containg `.`, the following strategy is used to try and access the routine:

1. Look for a `.yaml` files in the current directory with a name matching the label preceding the last `.`, with a routine matching the label following the last `.`.
2. Try to import a module using the the part of the label preceding the last `.`, with an importable object matching the label following the last `.`.

3.1.3 Additional Properties

Any additional properties will tell the routine about itself when created.

3.2 Running a Workflow

Run the whole thing: `timeflow workflow.yaml`

Run a routine (and all its dependencies): `timeflow workflow.yaml routine`

Specifiy what sort of output you want: `timeflow workflow -o csv`

- The label after the `-o` file will first try to use a builtin timeflow, or try to import an export routine if it contains a dot.