intake_parquet Documentation

Release 0.2.1

Joseph Crail

May 29, 2023

CONTENTS:

CHAPTER

ONE

QUICKSTART

`intake_parquet` provides quick and easy access to tabular data stored in the Apache Parquet binary, columnar format.

1.1 Installation

To use this plugin for intake, install with the following command:

```
conda install -c conda-forge intake-parquet
```

1.2 Usage

1.2.1 Ad-hoc

After installation, the function `intake.open_parquet` will become available. This can be used to open any parquet data-set. For example, assuming the part `'mydata.parquet'` contains parquet data in one or more files, the following will load it into an in-memory pandas dataframe:

```
import intake
source = intake.open_parquet('mydata.parquet')
dataframe = source.read()
```

Arguments to `open_parquet`:

- `urlpath` : the location of the data. This can be a single file, a list of specific files,

  or a directory containing parquet files (normally containing a `_metadata` index file). The URLs can be local files or, if using a protocol specifier such as `'s3://'`, a remote file location.

- `storage_options` : other parameters that are to be passed to the file-system

  implementation, in the case that a remote file-system is referenced in `urlpath`. For specifics, see the dask documentation.

- `columns` : Of the possible set of columns stored in the data, only those specified

  will be read; other data are not accessed at all. If not specified, loads all columns.

- `index` : Set the given column as the index of the resultant data-frame. If not given,

  a default index may be set, if the information is available in the metadata of the data-set; or no index if not. Can be set to `False` to prevent setting an index.

- `filters` : A list of filters to consider excluding the loading of some of the partitions

of the data. For example, if there is a column called `'value'`, a filter like `('value', '>', 5)`, then partitions which contain *no values* matching the filter will not be loaded, but partitions containing *at least one* value which passes the filter will be loaded.

- `engine` : 'fastparquet' or 'pyarrow'. Which backend to read with.
- `gather_statistics` : bool or None (default). Gather the statistics for each dataset partition. By default, this will only be done if the _metadata file is available. Otherwise, statistics will only be gathered if True, because the footer of every file will be parsed (which is very slow on some systems).
- see `dd.read_parquet()` for the other named parameters that can be passed through.

A source so defined will provide the usual methods such as `discover` and `read_partition`.

1.2.2 Creating Catalog Entries

To include in a catalog, entries must specify `driver: parquet`. The further arguments are exactly the same as for `open_parquet`. Commonly, the choice of which columns to load can be left to the end-user, by including it as a parameter.

1.2.3 Using a Catalog

Assuming a catalog file called `cat.yaml`, containing a parquet source `pdata`, one could load it into a dataframe as follows:

```
import intake
cat = intake.open_catalog('cat.yaml')
df = cat.pdata.read()
```

Parquet data-sets are inherently partitioned, and the partitions can be accessed in random order or iterated over.

Parquet data also plays well with Dask parallel processing, so the method `to_dask()` can be considered. Importantly, sub-selecting from the columns of the Dask data-frame will prevent unnecessary loading of the non-required columns even in the case where columns selection has not been included in the catalog entry user parameters.

1.2.4 Caching

Parquet data-sets can be singular, lists of files, or whole directory trees. The first two can be cached using the standard "files" type cache, but the latter requires "dir" type cachimg to capture the whole structure. An example may look like:

```
cache:
  - type: dir
    regex: '{{ CATALOG_DIR }}/split'
    argkey: urlpath
    depth: 4
```

Where the extra `depth` parameter indicates the number of directory levels that should be scanned.

CHAPTER

TWO

API REFERENCE

intake_parquet.source.ParquetSource(*args, ...)	Source to load parquet datasets.

class intake_parquet.source.**ParquetSource**(**args*, ***kwargs*)

Source to load parquet datasets.

Produces a dataframe.

A parquet dataset may be a single file, a set of files in a single directory or a nested set of directories containing data-files.

The implementation uses either fastparquet or pyarrow, select with the *engine*= kwarg.

Common keyword parameters accepted by this Source:

- **columns: list of str or None**
  column names to load. If None, loads all
- **filters: list of tuples**
  row-group level filtering; a tuple like `('x', '>', 1)` would mean that if a row-group has a maximum value less than 1 for the column `x`, then it will be skipped. Row-level filtering is *not* performed.
- **engine: 'fastparquet' or 'pyarrow'**
  Which backend to read with.
- see pd.read_parquet and dd.read_parquet() for the other named parameters that can be passed through.

Attributes

cache
cache_dirs
cat
classname
description
dtype
entry
gui
Source GUI, with parameter selection and plotting

has_been_persisted
hvplot
Returns a hvPlot object to provide a high-level plotting API.

is_persisted
plot
Returns a hvPlot object to provide a high-level plotting API.

plots
List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
`close`()	Close open resources corresponding to this data source.
`configure_new`(**kwargs)	Create a new instance of this source with altered arguments
`describe`()	Description from the entry spec
`discover`()	Open resource and populate the source attributes.
`export`(path, **kwargs)	Save this data for sharing with other people
`get`(**kwargs)	Create a new instance of this source with altered arguments
`persist`([ttl])	Save data from this source to local persistent storage
`read`()	Create single pandas dataframe from the whole data-set
`read_chunked`()	Return iterator over container fragments of data source
`read_partition`(i)	Return a part of the data corresponding to i-th partition.
`to_dask`()	Return a dask container for this data source
`to_spark`()	Produce Spark DataFrame equivalent
`yaml`()	Return YAML representation of this data-source

get_persisted	
set_cache_dir	

read()

Create single pandas dataframe from the whole data-set

to_dask()

Return a dask container for this data source

to_spark()

Produce Spark DataFrame equivalent

This will ignore all arguments except the urlpath, which will be directly interpreted by Spark. If you need to configure the storage, that must be done on the spark side.

This method requires intake-spark. See its documentation for how to set up a spark Session.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX