pandagg Documentation

Release 0.1

Léonard Binet

Mar 02, 2020

Contents

CHAPTER 1

User Guide

1.1 Introduction

Note: This is a work in progress. Some sections still need to be furnished.

About tree structure. About interactive objects.

1.2 Build Search query

The `Query` class allows multiple ways to declare and udpate an Elasticsearch query.

Let's explore the multiple ways we have to declare the following query:

```
>>> expected_query = {'bool': {'must': [
>>>     {'terms': {'genres': ['Action', 'Thriller']}},
>>>     {'range': {'rank': {'gte': 7}}},
>>>     {'nested': {
>>>         'path': 'roles',
>>>         'query': {'bool': {'must': [
>>>             {'term': {'roles.gender': {'value': 'F'}}},
>>>             {'term': {'roles.role': {'value': 'Reporter'}}}]}
>>>          }
>>>     }}
>>> ]}}
```

1.2.1 Pandagg DSL

Pandagg provides a DSL to declare this query in a quite similar fashion:

```
>>> from pandagg.query import Nested, Bool, Query, Range, Term, Terms
```

```
>>> q = Query(
>>>     Bool(must=[
>>>         Terms('genres', terms=['Action', 'Thriller']),
>>>         Range('rank', gte=7),
>>>         Nested(
>>>             path='roles',
>>>             query=Bool(must=[
>>>                 Term('roles.gender', value='F'),
>>>                 Term('roles.role', value='Reporter')
>>>             ])
>>>         )
>>>     ])
>>>)
```

The serialized query is then available with *query_dict* method:

```
>>> q.query_dict() == expected_query
True
```

A visual representation of the query helps to have a clearer view:

```
>>> q
<Query>
bool
└── must
    ├── nested
    │   ├── path="roles"
    │   └── query
    │       └── bool
    │           └── must
    │               ├── term, field=roles.gender, value="F"
    │               └── term, field=roles.role, value="Reporter"
    ├── range, field=rank, gte=7
    └── terms, field=genres, values=['Action', 'Thriller']
```

1.2.2 Chaining

Another way to declare this query is through chaining:

```
>>> from pandagg.utils import equal_queries
>>> from pandagg.query import Nested, Bool, Query, Range, Term, Terms
```

```
>>> q = Query()\
>>>     .query({'terms': {'genres': ['Action', 'Thriller']}})\
>>>     .nested(path='roles', _name='nested_roles', query=Term('roles.gender', value=
↪'F'))\
>>>     .query(Range('rank', gte=7))\
>>>     .query(Term('roles.role', value='Reporter'), parent='nested_roles')
```

```
>>> equal_queries(q.query_dict(), expected_query)
True
```

Note: *equal_queries* function won't consider order of clauses in must/should parameters since it actually doesn't matter in Elasticsearch execution, ie

```
>>> equal_queries({'must': [A, B]}, {'must': [B, A]})
True
```

1.2.3 Regular syntax

Eventually, you can also use regular Elasticsearch dict syntax:

```
>>> q = Query(expected_query)
>>> q
<Query>
bool
└── must
    ├── nested
    │   ├── path="roles"
    │   └── query
    │       └── bool
    │           └── must
    │               ├── term, field=roles.gender, value="F"
    │               └── term, field=roles.role, value="Reporter"
    ├── range, field=rank, gte=7
    └── terms, field=genres, values=['Action', 'Thriller']
```

1.3 Build Aggregation query

TODO

1.4 Parse Aggregation response

TODO

1.5 Explore your cluster indices

TODO

1.6 Navigate in a mapping

TODO

CHAPTER 2

Advanced usage

TODO

CHAPTER 3

Usage example on IMDB

An example based on publicly available IMDB data is documented in repository *examples/imdb* directory, with a jupyter notebook to showcase some of *pandagg* functionalities: here it is.

CHAPTER 4

pandagg package

4.1 Subpackages

4.1.1 pandagg.interactive package

4.1.1.1 Submodules

pandagg.interactive.abstract module

pandagg.interactive.client module

pandagg.interactive.index module

pandagg.interactive.mapping module

pandagg.interactive.response module

4.1.1.2 Module contents

4.1.2 pandagg.node package

4.1.2.1 Subpackages

pandagg.node.agg package

Submodules

pandagg.node.agg.abstract module

pandagg.tree.response module

4.1.3.2 Module contents

4.2 Submodules

4.2.1 pandagg.agg module

4.2.2 pandagg.client module

4.2.3 pandagg.exceptions module

4.2.4 pandagg.mapping module

4.2.5 pandagg.query module

4.2.6 pandagg.utils module

4.3 Module contents

CHAPTER 5

Contributing

TODO

pandagg is a Python package providing a simple interface to manipulate ElasticSearch queries and aggregations. It brings the following features:

- flexible aggregation and search queries declaration
- query validation based on provided mapping
- parsing of aggregation results in handy format: interactive bucket tree, normalized tree or tabular breakdown
- mapping interactive navigation

CHAPTER 6

Installing

pandagg can be installed with pip:

```
$ pip install pandagg
```

Alternatively, you can grab the latest source code from GitHub:

```
$ git clone git://github.com/alkemics/pandagg.git
$ python setup.py install
```

CHAPTER 7

Usage

The *User Guide* is the place to go to learn how to use the library and accomplish common tasks. The more in-depth *Advanced usage* guide is the place to go for deeply nested queries.

An example based on publicly available IMDB data is documented in repository *examples/imdb* directory, with a jupyter notebook to showcase some of *pandagg* functionalities: here it is.

The *pandagg package* documentation provides API-level documentation.

CHAPTER 8

License

pandagg is made available under the MIT License. For more details, see LICENSE.txt.

CHAPTER 9

Contributing

We happily welcome contributions, please see *Contributing* for details.