
pandagg Documentation

Release 0.1

Léonard Binet

Jun 22, 2020

1	Principles	1
1.1	Elasticsearch tree structures	1
1.2	Interactive usage	1
2	User Guide	3
2.1	Query	3
2.1.1	Instantiation	3
2.1.1.1	From native “dict” query	3
2.1.1.2	With DSL classes	4
2.1.1.3	With single clause as flattened syntax	5
2.1.2	Query enrichment	5
2.1.2.1	query() method	5
2.1.2.2	Compound clauses specific methods	6
2.1.2.3	Inserted clause location	7
2.2	Aggregation	8
2.2.1	Aggregation declaration	8
2.2.2	Aggregation response	8
2.3	Search	8
2.4	Mapping	8
2.4.1	Interactive mapping	8
2.5	Cluster indices discovery	9
3	Advanced usage	11
4	IMDB dataset	13
4.1	Query requirements	13
4.2	Data source	13
4.3	Index mapping	14
4.3.1	Overview	14
4.3.2	Which fields require nesting?	14
4.3.3	Text or keyword fields?	14
4.3.4	Mapping	14
4.4	Steps to start playing with your index	15
4.4.1	Dump tables	15
4.4.2	Clone pandagg and setup environment	15
4.4.3	Serialize movie documents and insert them	16
4.4.4	Explore pandagg notebooks	16

5	pandagg package	17
5.1	Subpackages	17
5.1.1	pandagg.interactive package	17
5.1.1.1	Submodules	17
5.1.1.2	Module contents	18
5.1.2	pandagg.node package	18
5.1.2.1	Subpackages	18
5.1.2.2	Submodules	37
5.1.2.3	Module contents	37
5.1.3	pandagg.tree package	37
5.1.3.1	Subpackages	37
5.1.3.2	Submodules	50
5.1.3.3	Module contents	52
5.2	Submodules	52
5.2.1	pandagg.aggs module	52
5.2.2	pandagg.connections module	58
5.2.3	pandagg.discovery module	59
5.2.4	pandagg.exceptions module	59
5.2.5	pandagg.mapping module	59
5.2.6	pandagg.query module	64
5.2.7	pandagg.response module	69
5.2.8	pandagg.search module	71
5.2.9	pandagg.utils module	77
5.3	Module contents	78
6	Contributing to Pandagg	79
6.1	Our Development Process	79
6.2	Pull Requests	79
6.3	Any contributions you make will be under the MIT Software License	79
6.4	Issues	80
6.5	Report bugs using Github's issues	80
6.6	Write bug reports with detail, background, and sample code	80
6.7	License	80
6.8	References	80
7	Installing	81
8	Usage	83
9	License	85
10	Contributing	87
	Python Module Index	89
	Index	91

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

This library focuses on two principles:

- stick to the **tree** structure of Elasticsearch objects
- provide simple and flexible interfaces to make it easy and intuitive to use in an interactive usage

1.1 Elasticsearch tree structures

Many Elasticsearch objects have a **tree** structure, ie they are built from a hierarchy of **nodes**:

- a **mapping** (tree) is a hierarchy of **fields** (nodes)
- a **query** (tree) is a hierarchy of query clauses (nodes)
- an **aggregation** (tree) is a hierarchy of aggregation clauses (nodes)
- an aggregation response (tree) is a hierarchy of response buckets (nodes)

This library sticks to that structure by providing a flexible syntax distinguishing **trees** and **nodes**, **trees** all inherit from `lighttree.Tree` class, whereas nodes all inherit from `lighttree.Node` class.

1.2 Interactive usage

pandagg is designed for both for “regular” code repository usage, and “interactive” usage (ipython or jupyter notebook usage with autocompletion features inspired by **pandas** design).

Some classes are not intended to be used elsewhere than in interactive mode (ipython), since their purpose is to serve auto-completion features and convenient representations.

Namely:

- *IMapping*: used to interactively navigate in mapping and run quick aggregations on some fields
- *IResponse*: used to interactively navigate in an aggregation response

These use case will be detailed in following sections.

Note: Examples will be based on *IMDB dataset* data. This is a work in progress. Some sections still need to be furnished.

2.1 Query

The *Query* class provides :

- multiple syntaxes to declare and update a query
- query validation (with nested clauses validation)
- ability to insert clauses at specific points
- tree-like visual representation

2.1.1 Instantiation

2.1.1.1 From native “dict” query

Given 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'}}}]}}  
>>>   ]}}  
>>>
```

(continues on next page)

(continued from previous page)

```
>>>     }}
>>> ]}}
```

To instantiate *Query*, simply pass “dict” query as argument:

```
>>> from pandagg.query import Query
>>> q = Query(expected_query)
```

A visual representation of the query is available with *show()*:

```
>>> q.show()
<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, genres=["Action", "Thriller"]
```

Call *to_dict()* to convert it to native dict:

```
>>> q.to_dict()
{'bool': {
  'must': [
    {'range': {'rank': {'gte': 7}}},
    {'terms': {'genres': ['Action', 'Thriller']}},
    {'bool': {'must': [
      {'term': {'roles.role': {'value': 'Reporter'}}},
      {'term': {'roles.gender': {'value': 'F'}}}]]}}
  ]
}}
```

```
>>> from pandagg.utils import equal_queries
>>> equal_queries(q.to_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
```

2.1.1.2 With DSL classes

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

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



```
>>> q = Bool(must=[
>>>     Terms(genres=['Action', 'Thriller']),
>>>     Range(rank={"gte": 7}),
>>>     Nested(
>>>         path='roles',
>>>         query=Bool(must=[
>>>             Term(roles__gender='F'),
>>>             Term(roles__role='Reporter')
>>>         ])
>>>     )
>>> ])
```

All these classes inherit from `Query` and thus provide the same interface.

```
>>> from pandagg.query import Query
>>> isinstance(q, Query)
True
```

2.1.1.3 With single clause as flattened syntax

In the flattened syntax, the query clause type is used as first argument:

```
>>> from pandagg.query import Query
>>> q = Query('terms', genres=['Action', 'Thriller'])
```

2.1.2 Query enrichment

All methods described below return a new `Query` instance, and keep unchanged the initial query.

For instance:

```
>>> from pandagg.query import Query
>>> initial_q = Query()
>>> enriched_q = initial_q.query('terms', genres=['Comedy', 'Short'])
```

```
>>> initial_q.to_dict()
None
```

```
>>> enriched_q.to_dict()
{'terms': {'genres': ['Comedy', 'Short']}}
```

Note: Calling `to_dict()` on an empty `Query` returns `None`

```
>>> from pandagg.query import Query
>>> Query().to_dict()
None
```

2.1.2.1 query() method

The base method to enrich a `Query` is `query()`.

Considering this query:

```
>>> from pandagg.query import Query
>>> q = Query()
```

`query()` accepts following syntaxes:

from dictionary:

```
>>> q.query({"terms": {"genres": ['Comedy', 'Short']}})
```

flattened syntax:

```
>>> q.query("terms", genres=['Comedy', 'Short'])
```

from Query instance (this includes DSL classes):

```
>>> from pandagg.query import Terms
>>> q.query(Terms(genres=['Action', 'Thriller']))
```

2.1.2.2 Compound clauses specific methods

`Query` instance also exposes following methods for specific compound queries:

(TODO: detail allowed syntaxes)

Specific to bool queries:

- `bool()`
- `filter()`
- `must()`
- `must_not()`
- `should()`

Specific to other compound queries:

- `nested()`
- `constant_score()`
- `dis_max()`
- `function_score()`
- `has_child()`
- `has_parent()`
- `parent_id()`
- `pinned_query()`
- `script_score()`
- `boost()`

2.1.2.3 Inserted clause location

On all insertion methods detailed above, by default, the inserted clause is placed at the top level of your query, and generates a bool clause if necessary.

Considering the following query:

```
>>> from pandagg.query import Query
>>> q = Query('terms', genres=['Action', 'Thriller'])
>>> q.show()
<Query>
terms, genres=["Action", "Thriller"]
```

A bool query will be created:

```
>>> q = q.query('range', rank={"gte": 7})
>>> q.show()
<Query>
bool
├─ must
│   └─ range, field=rank, gte=7
│       └─ terms, genres=["Action", "Thriller"]
```

And reused if necessary:

```
>>> q = q.must_not('range', year={"lte": 1970})
>>> q.show()
<Query>
bool
├─ must
│   └─ range, field=rank, gte=7
│       └─ terms, genres=["Action", "Thriller"]
└─ must_not
    └─ range, field=year, lte=1970
```

Specifying a specific location requires to name queries :

```
>>> from pandagg.query import Nested
```

```
>>> q = q.nested(path='roles', _name='nested_roles', query=Term('roles.gender', value=
↳ 'F'))
>>> q.show()
<Query>
bool
├─ must
│   └─ nested, _name=nested_roles, path="roles"
│       └─ query
│           └─ term, field=roles.gender, value="F"
│   └─ range, field=rank, gte=7
│       └─ terms, genres=["Action", "Thriller"]
└─ must_not
    └─ range, field=year, lte=1970
```

Doing so allows to insert clauses above/below given clause using *parent/child* parameters:

```
>>> q = q.query('term', roles__role='Reporter', parent='nested_roles')
>>> q.show()
```

(continues on next page)

(continued from previous page)

```

<Query>
bool
├── must
│   ├── nested, _name=nested_roles, path="roles"
│   │   └── query
│   │       └── bool
│   │           └── must
│   │               ├── term, field=roles.role, value="Reporter"
│   │               └── term, field=roles.gender, value="F"
│   ├── range, field=rank, gte=7
│   └── terms, genres=["Action", "Thriller"]
└── must_not
    └── range, field=year, lte=1970

```

TODO: explain *parent_param*, *child_param*, *mode* merging strategies on same named clause etc..

2.2 Aggregation

The *Aggs* class provides :

- multiple syntaxes to declare and update a aggregation
- clause validation (with nested clauses validation)
- ability to insert clauses at specific points

2.2.1 Aggregation declaration

2.2.2 Aggregation response

TODO

2.3 Search

TODO

2.4 Mapping

2.4.1 Interactive mapping

In interactive context, the *IMapping* class provides navigation features with autocompletion to quickly discover a large mapping:

```

>>> from pandagg.mapping import IMapping
>>> from examples.imdb.load import mapping
>>> m = IMapping(imdb_mapping)
>>> m.roles
<IMapping subpart: roles>
roles

```

[Nested]

(continues on next page)

(continued from previous page)

```

├── actor_id          Integer
├── first_name       Text
│   └── raw          ~ Keyword
├── gender           Keyword
├── last_name        Text
│   └── raw          ~ Keyword
└── role             Keyword

>>> m.roles.first_name
<IMapping subpart: roles.first_name>
first_name          Text
└── raw             ~ Keyword

```

To get the complete field definition, just call it:

```

>>> m.roles.first_name()
<Mapping Field first_name> of type text:
{
  "type": "text",
  "fields": {
    "raw": {
      "type": "keyword"
    }
  }
}

```

A **IMapping** instance can be bound to an Elasticsearch client to get quick access to aggregations computation on mapping fields.

Suppose you have the following client:

```

>>> from elasticsearch import Elasticsearch
>>> client = Elasticsearch(hosts=['localhost:9200'])

```

Client can be bound at instantiation:

```

>>> m = IMapping(imdb_mapping, client=client, index_name='movies')

```

Doing so will generate a **a** attribute on mapping fields, this attribute will list all available aggregation for that field type (with autocompletion):

```

>>> m.roles.gender.a.terms()
[('M', {'key': 'M', 'doc_count': 2296792}),
 ('F', {'key': 'F', 'doc_count': 1135174})]

```

Note: Nested clauses will be automatically taken into account.

2.5 Cluster indices discovery

TODO

CHAPTER 3

Advanced usage

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

- node and tree deserialization order
 - compound query insertion
-

You might know the Internet Movie Database, commonly called [IMDB](#).

Well it's a simple example to showcase some of Elasticsearch capabilities.

In this case, relational databases (SQL) are a good fit to store with consistence this kind of data. Yet indexing some of this data in a optimized search engine will allow more powerful queries.

4.1 Query requirements

In this example, we'll suppose most usage/queries requirements will be around the concept of movie (rather than usages focused on fetching actors or directors, even though it will still be possible with this data structure).

The index should provide good performances trying to answer these kind question (non-exhaustive):

- in which movies this actor played?
- what movies genres were most popular among decades?
- which actors have played in best-rated movies, or worst-rated movies?
- which actors movies directors prefer to cast in their movies?
- which are best ranked movies of last decade in Action or Documentary genres?
- ...

4.2 Data source

I exported following SQL tables from MariaDB [following these instructions](#).

Relational schema is the following:

imdb tables

4.3 Index mapping

4.3.1 Overview

The base unit (document) will be a movie, having a name, rank (ratings), year of release, a list of actors and a list of directors.

Schematically:

```
Movie:
- name
- year
- rank
- [] genres
- [] directors
- [] actor roles
```

4.3.2 Which fields require nesting?

Since genres contain a single keyword field, in no case we need it to be stored as a nested field. On the contrary, actor roles and directors require a nested mapping if we consider applying multiple simultaneous query clauses on their sub-fields (for instance search movie in which actor is a woman AND whose role is nurse). More information on distinction between array and nested fields [here](#).

4.3.3 Text or keyword fields?

Some fields are easy to choose, in no situation gender will require a full text search, thus we'll store it as a keyword. On the other hand actors and directors names (first and last) will require full-text search, we'll thus opt for a text field. Yet we might want to aggregate on exact keywords to count number of movies per actor for instance. More information on distinction between text and keyword fields [here](#)

4.3.4 Mapping

```
<Mapping>

-
├── directors
│   ├── director_id      [Nested]
│   ├── first_name      Keyword
│   │   └── raw          Text
│   ├── full_name       ~ Keyword
│   │   └── raw          Text
│   ├── genres          ~ Keyword
│   │   └── raw          Keyword
│   ├── last_name       Text
│   │   └── raw          ~ Keyword
├── genres               Keyword
├── movie_id             Keyword
├── name                 Text
│   └── raw              ~ Keyword
├── nb_directors         Integer
├── nb_roles             Integer
├── rank                 Float
└── roles                [Nested]
```

(continues on next page)

(continued from previous page)

— actor_id	Keyword
— first_name	Text
└─ raw	~ Keyword
— full_name	Text
└─ raw	~ Keyword
— gender	Keyword
— last_name	Text
└─ raw	~ Keyword
— role	Keyword
— year	Integer

4.4 Steps to start playing with your index

Note to Elastic, if you have a spare cluster to prepare demo indices on which you could let your community perform read operations we could skip this step ;)

4.4.1 Dump tables

Follow instruction on bottom of <https://relational.fit.cvut.cz/dataset/IMDb> page and dump following tables in a directory:

- movies.csv
- movies_genres.csv
- movies_directors.csv
- directors.csv
- directors_genres.csv
- roles.csv
- actors.csv

4.4.2 Clone pandagg and setup environment

```
git clone git@github.com:alkemics/pandagg.git
cd pandagg

virtualenv env
python setup.py develop
pip install pandas simplejson jupyter seaborn
```

Then copy `conf.py.dist` file into `conf.py` and edit variables as suits you, for instance:

```
# your cluster address
ES_HOST = 'localhost:9200'

# where your table dumps are stored, and where serialized output will be written
DATA_DIR = '/path/to/dumps/'
OUTPUT_FILE_NAME = 'serialized.json'
```

4.4.3 Serialize movie documents and insert them

```
# generate serialized movies documents, ready to be inserted in ES
# can take a while
python examples/imdb/serialize.py

# create index with mapping if necessary, bulk insert documents in ES
python examples/imdb/load.py
```

4.4.4 Explore pandagg notebooks

An example notebook is available to showcase some of pandagg functionalities: [here it is](#).

Code is present in `examples/imdb/IMDB_exploration.py` file.

5.1 Subpackages

5.1.1 pandagg.interactive package

5.1.1.1 Submodules

pandagg.interactive.mapping module

```
class pandagg.interactive.mapping.IMapping (*args, **kwargs)
    Bases: lighttree.interactive.TreeBasedObj

    Interactive wrapper upon mapping tree, allowing field navigation and quick access to single clause aggregations
    computation.
```

pandagg.interactive.response module

```
class pandagg.interactive.response.IResponse (tree, client=None, index_name=None,
                                              root_path=None, depth=None, initial_tree=None, query=None)

    Bases: lighttree.interactive.TreeBasedObj

    Interactive aggregation response.

    get_bucket_filter ()
        Build filters to select documents belonging to that bucket

    list_documents (**body)
        Return ES aggregation query to list documents belonging to given bucket. :return:
```

5.1.1.2 Module contents

5.1.2 pandagg.node package

5.1.2.1 Subpackages

pandagg.node.aggs package

Submodules

pandagg.node.aggs.abstract module

class pandagg.node.aggs.abstract.**AggNode** (*name, meta=None, **body*)

Bases: pandagg.node._node.Node

Wrapper around elasticsearch aggregation concept. <https://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations.html>

Each aggregation can be seen both a Node that can be encapsulated in a parent agg.

Define a method to build aggregation request.

BLACKLISTED_MAPPING_TYPES = None

KEY = None

VALUE_ATTRS = None

WHITELISTED_MAPPING_TYPES = None

classmethod **extract_bucket_value** (*response, value_as_dict=False*)

extract_buckets (*response_value*)

get_filter (*key*)

Return filter query to list documents having this aggregation key. :param key: string :return: elasticsearch filter query

line_repr (*depth, **kwargs*)

Control how node is displayed in tree representation.

to_dict (*with_name=False*)

ElasticSearch aggregation queries follow this formatting:

```
{
  "<aggregation_name>" : {
    "<aggregation_type>" : {
      <aggregation_body>
    }
    [, "meta" : {   [<meta_data_body>] } ]?
  }
}
```

Query dict returns the following part (without aggregation name):

```
{
  "<aggregation_type>" : {
    <aggregation_body>
  }
}
```

(continues on next page)

(continued from previous page)

```
[, "meta" : { [<meta_data_body> ] } ]?
}
```

classmethod `valid_on_field_type` (*field_type*)

class `pandagg.node.aggs.abstract.BucketAggNode` (*name*, *meta=None*, ***body*)

Bases: `pandagg.node.aggs.abstract.AggNode`

Bucket aggregation have special abilities: they can encapsulate other aggregations as children. Each time, the extracted value is a 'doc_count'.

Provide methods: - to build aggregation request (with children aggregations) - to to extract buckets from raw response - to build query to filter documents belonging to that bucket

Note: the aggs attribute's only purpose is for children initiation with the following syntax: >>> from pandagg.aggs import Terms, Avg >>> agg = Terms(>>> name='term_agg', >>> field='some_path', >>> aggs=[>>> Avg(agg_name='avg_agg', field='some_other_path') >>>] >>>)

VALUE_ATTRS = None

extract_buckets (*response_value*)

get_filter (*key*)

Provide filter to get documents belonging to document of given key.

class `pandagg.node.aggs.abstract.FieldOrScriptMetricAgg` (*name*, *meta=None*, ***body*)

Bases: `pandagg.node.aggs.abstract.MetricAgg`

Metric aggregation based on single field.

VALUE_ATTRS = None

class `pandagg.node.aggs.abstract.MetricAgg` (*name*, *meta=None*, ***body*)

Bases: `pandagg.node.aggs.abstract.AggNode`

Metric aggregation are aggregations providing a single bucket, with value attributes to be extracted.

VALUE_ATTRS = None

extract_buckets (*response_value*)

get_filter (*key*)

Return filter query to list documents having this aggregation key. :param key: string :return: elasticsearch filter query

class `pandagg.node.aggs.abstract.MultipleBucketAgg` (*name*, *keyed=None*, *key_path='key'*, *meta=None*, ***body*)

Bases: `pandagg.node.aggs.abstract.BucketAggNode`

IMPLICIT_KEYED = False

VALUE_ATTRS = None

extract_buckets (*response_value*)

get_filter (*key*)

Provide filter to get documents belonging to document of given key.

class `pandagg.node.aggs.abstract.Pipeline` (*name*, *buckets_path*, *gap_policy=None*, *meta=None*, ***body*)

Bases: `pandagg.node.aggs.abstract.UniqueBucketAgg`

```
VALUE_ATTRS = None

get_filter(key)
    Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.abstract.ScriptPipeline(name, script, buckets_path,
                                                gap_policy=None, meta=None,
                                                **body)

    Bases: pandagg.node.aggs.abstract.Pipeline

    KEY = None

    VALUE_ATTRS = 'value'

class pandagg.node.aggs.abstract.ShadowRoot
    Bases: pandagg.node.aggs.abstract.UniqueBucketAgg

    Not a real aggregation.

    KEY = 'shadow_root'

    classmethod extract_bucket_value(response, value_as_dict=False)

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

    line_repr(depth, **kwargs)
        Control how node is displayed in tree representation.

class pandagg.node.aggs.abstract.UniqueBucketAgg(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.BucketAggNode

    Aggregations providing a single bucket.

    VALUE_ATTRS = None

    extract_buckets(response_value)

    get_filter(key)
        Provide filter to get documents belonging to document of given key.
```

pandagg.node.aggs.bucket module

Not implemented aggregations include: - children agg - geo-distance - geo-hash grid - ipv4 - sampler - significant terms

```
class pandagg.node.aggs.bucket.Composite(name, keyed=None, key_path='key', meta=None,
                                         **body)
    Bases: pandagg.node.aggs.abstract.MultipleBucketAgg

    KEY = 'composite'

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.DateHistogram(name, field, interval=None, calendar_interval=None,
                                              fixed_interval=None, meta=None, keyed=False,
                                              key_as_string=True, **body)
    Bases: pandagg.node.aggs.abstract.MultipleBucketAgg

    KEY = 'date_histogram'

    VALUE_ATTRS = ['doc_count']
```



```

    WHITELISTED_MAPPING_TYPES = ['date']

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.DateRange(name, field, key_as_string=True, meta=None,
                                          **body)
    Bases: pandagg.node.aggs.bucket.Range
    KEY = 'date_range'
    KEY_SEP = ':'
    VALUE_ATTRS = ['doc_count']
    WHITELISTED_MAPPING_TYPES = ['date']

class pandagg.node.aggs.bucket.Filter(name, filter=None, meta=None, **kwargs)
    Bases: pandagg.node.aggs.abstract.UniqueBucketAgg
    KEY = 'filter'
    VALUE_ATTRS = ['doc_count']
    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.Filters(name, filters, other_bucket=False,
                                       other_bucket_key=None, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.MultipleBucketAgg
    DEFAULT_OTHER_KEY = '_other_'
    IMPLICIT_KEYED = True
    KEY = 'filters'
    VALUE_ATTRS = ['doc_count']
    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.Global(name, meta=None)
    Bases: pandagg.node.aggs.abstract.UniqueBucketAgg
    KEY = 'global'
    VALUE_ATTRS = ['doc_count']
    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.Histogram(name, field, interval, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.MultipleBucketAgg
    KEY = 'histogram'
    VALUE_ATTRS = ['doc_count']
    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.MatchAll(name, meta=None)
    Bases: pandagg.node.aggs.bucket.Filter

```

```
class pandagg.node.aggs.bucket.Missing(name, field, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.UniqueBucketAgg

    BLACKLISTED_MAPPING_TYPES = []

    KEY = 'missing'

    VALUE_ATTRS = ['doc_count']

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.Nested(name, path, meta=None)
    Bases: pandagg.node.aggs.abstract.UniqueBucketAgg

    KEY = 'nested'

    VALUE_ATTRS = ['doc_count']

    WHITELISTED_MAPPING_TYPES = ['nested']

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.Range(name, field, ranges, keyed=False, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.MultipleBucketAgg

    KEY = 'range'

    KEY_SEP = '-'

    VALUE_ATTRS = ['doc_count']

    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

    from_key

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

    to_key

class pandagg.node.aggs.bucket.ReverseNested(name, path=None, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.UniqueBucketAgg

    KEY = 'reverse_nested'

    VALUE_ATTRS = ['doc_count']

    WHITELISTED_MAPPING_TYPES = ['nested']

    get_filter(key)
        Provide filter to get documents belonging to document of given key.

class pandagg.node.aggs.bucket.Terms(name, field, missing=None, size=None, meta=None,
                                     **body)
    Bases: pandagg.node.aggs.abstract.MultipleBucketAgg

    Terms aggregation.

    BLACKLISTED_MAPPING_TYPES = []

    KEY = 'terms'

    VALUE_ATTRS = ['doc_count', 'doc_count_error_upper_bound', 'sum_other_doc_count']

    get_filter(key)
        Provide filter to get documents belonging to document of given key.
```

pandagg.node.aggs.metric module

```

class pandagg.node.aggs.metric.Avg(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'avg'

    VALUE_ATTRS = ['value']

    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.Cardinality(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'cardinality'

    VALUE_ATTRS = ['value']

class pandagg.node.aggs.metric.ExtendedStats(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'extended_stats'

    VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum', 'sum_of_squares', 'variance', 'std

    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.GeoBound(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'geo_bounds'

    VALUE_ATTRS = ['bounds']

    WHITELISTED_MAPPING_TYPES = ['geo_point']

class pandagg.node.aggs.metric.GeoCentroid(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'geo_centroid'

    VALUE_ATTRS = ['location']

    WHITELISTED_MAPPING_TYPES = ['geo_point']

class pandagg.node.aggs.metric.Max(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'max'

    VALUE_ATTRS = ['value']

    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.Min(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'min'

    VALUE_ATTRS = ['value']

    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.PercentileRanks(name, field, values, meta=None,
                                                **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg

    KEY = 'percentile_ranks'

```

```
VALUE_ATTRS = ['values']
WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.Percentiles(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg
    Percents body argument can be passed to specify which percentiles to fetch.
    KEY = 'percentiles'
    VALUE_ATTRS = ['values']
    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.Stats(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg
    KEY = 'stats'
    VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum']
    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.Sum(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg
    KEY = 'sum'
    VALUE_ATTRS = ['value']
    WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

class pandagg.node.aggs.metric.TopHits(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.MetricAgg
    KEY = 'top_hits'
    VALUE_ATTRS = ['hits']

class pandagg.node.aggs.metric.ValueCount(name, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.FieldOrScriptMetricAgg
    BLACKLISTED_MAPPING_TYPES = []
    KEY = 'value_count'
    VALUE_ATTRS = ['value']
```

pandagg.node.aggs.pipeline module

Pipeline aggregations: <https://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-pipeline.html>

```
class pandagg.node.aggs.pipeline.AvgBucket(name, buckets_path, gap_policy=None,
                                           meta=None, **body)
    Bases: pandagg.node.aggs.abstract.Pipeline
    KEY = 'avg_bucket'
    VALUE_ATTRS = ['value']

class pandagg.node.aggs.pipeline.BucketScript(name, script, buckets_path,
                                              gap_policy=None, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.ScriptPipeline
    KEY = 'bucket_script'
```

```

    VALUE_ATTRS = ['value']

class pandagg.node.aggs.pipeline.BucketSelector(name, script, buckets_path,
                                                gap_policy=None, meta=None,
                                                **body)
    Bases: pandagg.node.aggs.abstract.ScriptPipeline
    KEY = 'bucket_selector'
    VALUE_ATTRS = None

class pandagg.node.aggs.pipeline.BucketSort(name, script, buckets_path,
                                              gap_policy=None, meta=None, **body)
    Bases: pandagg.node.aggs.abstract.ScriptPipeline
    KEY = 'bucket_sort'
    VALUE_ATTRS = None

class pandagg.node.aggs.pipeline.CumulativeSum(name, buckets_path, gap_policy=None,
                                                meta=None, **body)
    Bases: pandagg.node.aggs.abstract.Pipeline
    KEY = 'cumulative_sum'
    VALUE_ATTRS = ['value']

class pandagg.node.aggs.pipeline.Derivative(name, buckets_path, gap_policy=None,
                                             meta=None, **body)
    Bases: pandagg.node.aggs.abstract.Pipeline
    KEY = 'derivative'
    VALUE_ATTRS = ['value']

class pandagg.node.aggs.pipeline.ExtendedStatsBucket(name, buckets_path,
                                                       gap_policy=None, meta=None,
                                                       **body)
    Bases: pandagg.node.aggs.abstract.Pipeline
    KEY = 'extended_stats_bucket'
    VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum', 'sum_of_squares', 'variance', 'std']

class pandagg.node.aggs.pipeline.MaxBucket(name, buckets_path, gap_policy=None,
                                           meta=None, **body)
    Bases: pandagg.node.aggs.abstract.Pipeline
    KEY = 'max_bucket'
    VALUE_ATTRS = ['value']

class pandagg.node.aggs.pipeline.MinBucket(name, buckets_path, gap_policy=None,
                                           meta=None, **body)
    Bases: pandagg.node.aggs.abstract.Pipeline
    KEY = 'min_bucket'
    VALUE_ATTRS = ['value']

class pandagg.node.aggs.pipeline.MovingAvg(name, buckets_path, gap_policy=None,
                                             meta=None, **body)
    Bases: pandagg.node.aggs.abstract.Pipeline
    KEY = 'moving_avg'
    VALUE_ATTRS = ['value']

```

```
class pandagg.node.aggs.pipeline.PercentilesBucket(name, buckets_path,
                                                    gap_policy=None, meta=None,
                                                    **body)

    Bases: pandagg.node.aggs.abstract.Pipeline

    KEY = 'percentiles_bucket'

    VALUE_ATTRS = ['values']

class pandagg.node.aggs.pipeline.SerialDiff(name, buckets_path, gap_policy=None,
                                              meta=None, **body)

    Bases: pandagg.node.aggs.abstract.Pipeline

    KEY = 'serial_diff'

    VALUE_ATTRS = ['value']

class pandagg.node.aggs.pipeline.StatsBucket(name, buckets_path, gap_policy=None,
                                              meta=None, **body)

    Bases: pandagg.node.aggs.abstract.Pipeline

    KEY = 'stats_bucket'

    VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum']

class pandagg.node.aggs.pipeline.SumBucket(name, buckets_path, gap_policy=None,
                                             meta=None, **body)

    Bases: pandagg.node.aggs.abstract.Pipeline

    KEY = 'sum_bucket'

    VALUE_ATTRS = ['value']
```

Module contents

pandagg.node.mapping package

Submodules

pandagg.node.mapping.abstract module

```
class pandagg.node.mapping.abstract.Field(name, key, **body)
    Bases: pandagg.node._node.Node

    body

    line_repr(depth, **kwargs)
        Control how node is displayed in tree representation.

class pandagg.node.mapping.abstract.ShadowRoot(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedComplexField

    KEY = '_'

class pandagg.node.mapping.abstract.UnnamedComplexField(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    KEY = None

class pandagg.node.mapping.abstract.UnnamedField(**body)
    Bases: object
```

```

    KEY = None

    classmethod get_dsl_class (name)

    to_named_field (name, _subfield=False)

class pandagg.node.mapping.abstract.UnnamedRegularField (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    KEY = None

```

pandagg.node.mapping.field_datatypes module

<https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html>

```

class pandagg.node.mapping.field_datatypes.Alias (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Defines an alias to an existing field.

    KEY = 'alias'

class pandagg.node.mapping.field_datatypes.Binary (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'binary'

class pandagg.node.mapping.field_datatypes.Boolean (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'boolean'

class pandagg.node.mapping.field_datatypes.Byte (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'byte'

class pandagg.node.mapping.field_datatypes.Completion (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    To provide auto-complete suggestions

    KEY = 'completion'

class pandagg.node.mapping.field_datatypes.Date (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'date'

class pandagg.node.mapping.field_datatypes.DateNanos (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'date_nanos'

class pandagg.node.mapping.field_datatypes.DateRange (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'date_range'

class pandagg.node.mapping.field_datatypes.DenseVector (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Record dense vectors of float values.

    KEY = 'dense_vector'

```

```
class pandagg.node.mapping.field_datatypes.Double(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'double'

class pandagg.node.mapping.field_datatypes.DoubleRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'double_range'

class pandagg.node.mapping.field_datatypes.Flattened(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Allows an entire JSON object to be indexed as a single field.

    KEY = 'flattened'

class pandagg.node.mapping.field_datatypes.Float(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'float'

class pandagg.node.mapping.field_datatypes.FloatRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'float_range'

class pandagg.node.mapping.field_datatypes.GeoPoint(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    For lat/lon points

    KEY = 'geo_point'

class pandagg.node.mapping.field_datatypes.GeoShape(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    For complex shapes like polygons

    KEY = 'geo_shape'

class pandagg.node.mapping.field_datatypes.HalfFloat(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'half_float'

class pandagg.node.mapping.field_datatypes.Histogram(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    For pre-aggregated numerical values for percentiles aggregations.

    KEY = 'histogram'

class pandagg.node.mapping.field_datatypes.IP(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    for IPv4 and IPv6 addresses

    KEY = 'IP'

class pandagg.node.mapping.field_datatypes.Integer(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'integer'

class pandagg.node.mapping.field_datatypes.IntegerRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField
```



```

    KEY = 'integer_range'

class pandagg.node.mapping.field_datatypes.Join(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Defines parent/child relation for documents within the same index

    KEY = 'join'

class pandagg.node.mapping.field_datatypes.Keyword(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'keyword'

class pandagg.node.mapping.field_datatypes.Long(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'long'

class pandagg.node.mapping.field_datatypes.LongRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'long_range'

class pandagg.node.mapping.field_datatypes.MapperAnnotatedText(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    To index text containing special markup (typically used for identifying named entities)

    KEY = 'annotated-text'

class pandagg.node.mapping.field_datatypes.MapperMurMur3(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    To compute hashes of values at index-time and store them in the index

    KEY = 'murmur3'

class pandagg.node.mapping.field_datatypes.Nested(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedComplexField

    KEY = 'nested'

class pandagg.node.mapping.field_datatypes.Object(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedComplexField

    KEY = 'object'

class pandagg.node.mapping.field_datatypes.Percolator(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Accepts queries from the query-dsl

    KEY = 'percolator'

class pandagg.node.mapping.field_datatypes.RankFeature(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Record numeric feature to boost hits at query time.

    KEY = 'rank_feature'

class pandagg.node.mapping.field_datatypes.RankFeatures(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Record numeric features to boost hits at query time.

    KEY = 'rank_features'

```

```
class pandagg.node.mapping.field_datatypes.ScaledFloat (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'scaled_float'

class pandagg.node.mapping.field_datatypes.SearchAsYouType (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    A text-like field optimized for queries to implement as-you-type completion

    KEY = 'search_as_you_type'

class pandagg.node.mapping.field_datatypes.Shape (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    For arbitrary cartesian geometries.

    KEY = 'shape'

class pandagg.node.mapping.field_datatypes.Short (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'short'

class pandagg.node.mapping.field_datatypes.SparseVector (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    Record sparse vectors of float values.

    KEY = 'sparse_vector'

class pandagg.node.mapping.field_datatypes.Text (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'text'

class pandagg.node.mapping.field_datatypes.TokenCount (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    To count the number of tokens in a string

    KEY = 'token_count'
```

pandagg.node.mapping.meta_fields module

```
class pandagg.node.mapping.meta_fields.FieldNames (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    All fields in the document which contain non-null values.

    KEY = '_field_names'

class pandagg.node.mapping.meta_fields.Id (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    The document's ID.

    KEY = '_id'

class pandagg.node.mapping.meta_fields.Ignored (**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    All fields in the document that have been ignored at index time because of ignore_malformed.

    KEY = '_ignored'
```

```
class pandagg.node.mapping.meta_fields.Index(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    The index to which the document belongs.

    KEY = '_index'

class pandagg.node.mapping.meta_fields.Meta(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    Application specific metadata.

    KEY = '_meta'

class pandagg.node.mapping.meta_fields.Routing(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    A custom routing value which routes a document to a particular shard.

    KEY = '_routing'

class pandagg.node.mapping.meta_fields.Size(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    The size of the _source field in bytes, provided by the mapper-size plugin.

    KEY = '_size'

class pandagg.node.mapping.meta_fields.Source(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    The original JSON representing the body of the document.

    KEY = '_source'

class pandagg.node.mapping.meta_fields.Type(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField

    The document's mapping type.

    KEY = '_type'
```

Module contents

pandagg.node.query package

Submodules

pandagg.node.query.abstract module

```
class pandagg.node.query.abstract.AbstractSingleFieldQueryClause(field,
                                                                    _name=None,
                                                                    **body)
    Bases: pandagg.node.query.abstract.LeafQueryClause

class pandagg.node.query.abstract.FlatFieldQueryClause(field, _name=None,
                                                         **body)
    Bases: pandagg.node.query.abstract.AbstractSingleFieldQueryClause

    Query clause applied on one single field. Example:

    Exists: {"exists": {"field": "user"}} -> field = "user" -> body = {"field": "user"} q = Exists(field="user")
```

DistanceFeature: {"distance_feature": {"field": "production_date", "pivot": "7d", "origin": "now"}} -> field = "production_date" -> body = {"field": "production_date", "pivot": "7d", "origin": "now"} q = DistanceFeature(field="production_date", pivot="7d", origin="now")

```
class pandagg.node.query.abstract.KeyFieldQueryClause (field=None, _name=None,
                                                    _expand__to_dot=True,
                                                    **params)
```

Bases: [pandagg.node.query.abstract.AbstractSingleFieldQueryClause](#)

Clause with field used as key in clause body:

Term: {"term": {"user": {"value": "Kimchy", "boost": 1}}} -> field = "user" -> body = {"user": {"value": "Kimchy", "boost": 1}} q1 = Term(user={"value": "Kimchy", "boost": 1}) q2 = Term(field="user", value="Kimchy", boost=1)}

Can accept a "_implicit_param" attribute specifying which is the equivalent key when inner body isn't a dict but a raw value. For Term: _implicit_param = "value" q = Term(user="Kimchy") {"term": {"user": {"value": "Kimchy"}}} -> field = "user" -> body = {"term": {"user": {"value": "Kimchy"}}}

```
line_repr (depth, **kwargs)
```

Control how node is displayed in tree representation.

```
class pandagg.node.query.abstract.LeafQueryClause (**body)
```

Bases: [pandagg.node.query.abstract.QueryClause](#)

```
class pandagg.node.query.abstract.MultiFieldsQueryClause (fields, _name=None,
                                                         **body)
```

Bases: [pandagg.node.query.abstract.LeafQueryClause](#)

```
line_repr (depth, **kwargs)
```

Control how node is displayed in tree representation.

```
class pandagg.node.query.abstract.ParentParameterClause (**body)
```

Bases: [pandagg.node.query.abstract.QueryClause](#)

MULTIPLE = False

```
class pandagg.node.query.abstract.QueryClause (**body)
```

Bases: [pandagg.node._node.Node](#)

KEY = None

```
line_repr (depth, **kwargs)
```

Control how node is displayed in tree representation.

name

```
to_dict (with_name=True)
```

pandagg.node.query.compound module

```
class pandagg.node.query.compound.Bool (**body)
```

Bases: [pandagg.node.query.compound.CompoundClause](#)

KEY = 'bool'

```
class pandagg.node.query.compound.Boosting (**body)
```

Bases: [pandagg.node.query.compound.CompoundClause](#)

KEY = 'boosting'

```
class pandagg.node.query.compound.CompoundClause (**body)
    Bases: pandagg.node.query.abstract.QueryClause
```

Compound clauses can encapsulate other query clauses:

```
classmethod operator (key)
```

```
class pandagg.node.query.compound.ConstantScore (**body)
    Bases: pandagg.node.query.compound.CompoundClause
```

```
KEY = 'constant_score'
```

```
class pandagg.node.query.compound.DisMax (**body)
    Bases: pandagg.node.query.compound.CompoundClause
```

```
KEY = 'dis_max'
```

```
class pandagg.node.query.compound.FunctionScore (**body)
    Bases: pandagg.node.query.compound.CompoundClause
```

```
KEY = 'function_score'
```

pandagg.node.query.full_text module

```
class pandagg.node.query.full_text.Common (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause
```

```
KEY = 'common'
```

```
class pandagg.node.query.full_text.Intervals (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause
```

```
KEY = 'intervals'
```

```
class pandagg.node.query.full_text.Match (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause
```

```
KEY = 'match'
```

```
class pandagg.node.query.full_text.MatchBoolPrefix (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause
```

```
KEY = 'match_bool_prefix'
```

```
class pandagg.node.query.full_text.MatchPhrase (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause
```

```
KEY = 'match_phrase'
```

```
class pandagg.node.query.full_text.MatchPhrasePrefix (field=None, _name=None,
                                         _expand__to_dot=True,
                                         **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause
```

```
KEY = 'match_phrase_prefix'
```

```
class pandagg.node.query.full_text.MultiMatch (fields, _name=None, **body)
    Bases: pandagg.node.query.abstract.MultiFieldsQueryClause
```

```
KEY = 'multi_match'
```

```
class pandagg.node.query.full_text.QueryString(**body)
    Bases: pandagg.node.query.abstract.LeafQueryClause

    KEY = 'query_string'

class pandagg.node.query.full_text.SimpleQueryString(**body)
    Bases: pandagg.node.query.abstract.LeafQueryClause

    KEY = 'simple_string'
```

pandagg.node.query.geo module

```
class pandagg.node.query.geo.GeoBoundingBox(field=None, _name=None, _ex-
    pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

    KEY = 'geo_bounding_box'

class pandagg.node.query.geo.GeoDistance(distance, **body)
    Bases: pandagg.node.query.abstract.AbstractSingleFieldQueryClause

    KEY = 'geo_distance'

    line_repr(depth, **kwargs)
        Control how node is displayed in tree representation.

class pandagg.node.query.geo.GeoPolygone(field=None, _name=None, _ex-
    pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

    KEY = 'geo_polygon'

class pandagg.node.query.geo.GeoShape(field=None, _name=None, _expand__to_dot=True,
    **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

    KEY = 'geo_shape'
```

pandagg.node.query.joining module

```
class pandagg.node.query.joining.HasChild(**body)
    Bases: pandagg.node.query.compound.CompoundClause

    KEY = 'has_child'

class pandagg.node.query.joining.HasParent(**body)
    Bases: pandagg.node.query.compound.CompoundClause

    KEY = 'has_parent'

class pandagg.node.query.joining.Nested(path, **kwargs)
    Bases: pandagg.node.query.compound.CompoundClause

    KEY = 'nested'

class pandagg.node.query.joining.ParentId(**body)
    Bases: pandagg.node.query.abstract.LeafQueryClause

    KEY = 'parent_id'
```

pandagg.node.query.shape module

```
class pandagg.node.query.shape.Shape (**body)
    Bases: pandagg.node.query.abstract.LeafQueryClause
    KEY = 'shape'
```

pandagg.node.query.span module

pandagg.node.query.specialized module

```
class pandagg.node.query.specialized.DistanceFeature (field, _name=None, **body)
    Bases: pandagg.node.query.abstract.FlatFieldQueryClause
    KEY = 'distance_feature'

class pandagg.node.query.specialized.MoreLikeThis (fields, _name=None, **body)
    Bases: pandagg.node.query.abstract.MultiFieldsQueryClause
    KEY = 'more_like_this'

class pandagg.node.query.specialized.Percolate (field, _name=None, **body)
    Bases: pandagg.node.query.abstract.FlatFieldQueryClause
    KEY = 'percolate'

class pandagg.node.query.specialized.RankFeature (field, _name=None, **body)
    Bases: pandagg.node.query.abstract.FlatFieldQueryClause
    KEY = 'rank_feature'

class pandagg.node.query.specialized.Script (**body)
    Bases: pandagg.node.query.abstract.LeafQueryClause
    KEY = 'script'

class pandagg.node.query.specialized.Wrapper (**body)
    Bases: pandagg.node.query.abstract.LeafQueryClause
    KEY = 'wrapper'
```

pandagg.node.query.specialized_compound module

```
class pandagg.node.query.specialized_compound.PinnedQuery (**body)
    Bases: pandagg.node.query.compound.CompoundClause
    KEY = 'pinned'

class pandagg.node.query.specialized_compound.ScriptScore (**body)
    Bases: pandagg.node.query.compound.CompoundClause
    KEY = 'script_score'
```

pandagg.node.query.term_level module

```
class pandagg.node.query.term_level.Exists (field, _name=None)
    Bases: pandagg.node.query.abstract.LeafQueryClause
```

```
KEY = 'exists'

line_repr (depth, **kwargs)
    Control how node is displayed in tree representation.

class pandagg.node.query.term_level.Fuzzy (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'fuzzy'

class pandagg.node.query.term_level.Ids (values, _name=None)
    Bases: pandagg.node.query.abstract.LeafQueryClause

KEY = 'ids'

line_repr (depth, **kwargs)
    Control how node is displayed in tree representation.

to_dict (with_name=True)

class pandagg.node.query.term_level.Prefix (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'prefix'

class pandagg.node.query.term_level.Range (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'range'

class pandagg.node.query.term_level.Regexp (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'regexp'

class pandagg.node.query.term_level.Term (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'term'

class pandagg.node.query.term_level.Terms (**body)
    Bases: pandagg.node.query.abstract.AbstractSingleFieldQueryClause

KEY = 'terms'

class pandagg.node.query.term_level.TermsSet (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'terms_set'

class pandagg.node.query.term_level.Type (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'type'

class pandagg.node.query.term_level.Wildcard (field=None, _name=None, _ex-
                                         pand__to_dot=True, **params)
    Bases: pandagg.node.query.abstract.KeyFieldQueryClause

KEY = 'wildcard'
```


Module contents

pandagg.node.response package

Submodules

pandagg.node.response.bucket module

```
class pandagg.node.response.bucket.Bucket (value, key=None, level=None)
    Bases: pandagg.node._node.Node

    ROOT_NAME = 'root'

    attr_name
        Determine under which attribute name the bucket will be available in response tree. Dots are replaced by
        _ characters so that they don't prevent from accessing as attribute.

        Resulting attribute unfit for python attribute name syntax is still possible and will be accessible through
        item access (dict like), see more in 'utils.Obj' for more details.

    line_repr (**kwargs)
        Control how node is displayed in tree representation.
```

Module contents

5.1.2.2 Submodules

pandagg.node.types module

5.1.2.3 Module contents

5.1.3 pandagg.tree package

5.1.3.1 Subpackages

pandagg.tree.aggs package

Submodules

pandagg.tree.aggs.aggs module

```
class pandagg.tree.aggs.aggs.AbstractLeafAgg (*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.Aggs

    KEY = None
    Allow following syntax:
```

```
>>> a = Avg("my_terms_agg", field="yolo")
```

```
class pandagg.tree.aggs.aggs.AbstractParentAgg (*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.Aggs
```

KEY = None

Allow following syntax:

```
>>> a = Terms("my_terms_agg", field="yolo", aggs={...})
```

class pandagg.tree.aggs.aggs.**Aggs** (*args, **kwargs)

Bases: pandagg.tree._tree.Tree

Combination of aggregation clauses. This class provides handful methods to build an aggregation (see `aggs()` and `groupby()`), and is used as well to parse aggregations response in handy formats.

Mapping declaration is optional, but doing so validates aggregation validity and automatically handles missing nested clauses.

All following syntaxes are identical:

From a dict:

```
>>> Aggs({"per_user":{"terms":{"field":"user"}}})
```

Using shortcut declaration: first argument is the aggregation type, other arguments are aggregation body parameters:

```
>>> Aggs('terms', name='per_user', field='user')
```

Using DSL class:

```
>>> from pandagg.aggs import Terms
>>> Aggs(Terms('per_user', field='user'))
```

Dict and DSL class syntaxes allow to provide multiple clauses aggregations:

```
>>> Aggs({"per_user":{"terms":{"field":"user"}, "aggs": {"avg_age": {"avg": {
↪ "field": "age"}}}}})
```

Which is similar to:

```
>>> from pandagg.aggs import Terms, Avg
>>> Terms('per_user', field='user', aggs=Avg('avg_age', field='age'))
```

Keyword Arguments

- *mapping* (dict or `pandagg.tree.mapping.Mapping`) – Mapping of requested indice(s). Providing it will validate aggregations validity, and add required nested clauses if missing.
- *nested_autocorrect* (bool) – In case of missing nested clauses in aggregation, if True, automatically add missing nested clauses, else raise error.
- remaining kwargs: Used as body in aggregation

aggs (*args, **kwargs)

Arrange passed aggregations “horizontally”.

Given the initial aggregation:

```
A—> B
└─> C
```

If passing multiple aggregations with *insert_below* = ‘A’:

```

A—> B
└─> C
    └─> new1
        └─> new2

```

Note: those will be placed under the *insert_below* aggregation clause id if provided, else under the deepest linear bucket aggregation if there is no ambiguity:

OK:

```

A—> B -> C -> new

```

KO:

```

A—> B
└─> C

```

args accepts single occurrence or sequence of following formats:

- string (for terms agg concise declaration)
- regular Elasticsearch dict syntax
- AggNode instance (for instance Terms, Filters etc)

Keyword Arguments

- *insert_below* (string) – Parent aggregation name under which these aggregations should be placed
- *at_root* (string) – Insert aggregations at root of aggregation query
- remaining kwargs: Used as body in aggregation

Return type *pandagg.aggs.Aggs*

applied_nested_path_at_node (*nid*)

deepest_linear_bucket_agg

Return deepest bucket aggregation node (*pandagg.nodes.abstract.BucketAggNode*) of that aggregation that neither has siblings, nor has an ancestor with siblings.

groupby (**args, **kwargs*)

Arrange passed aggregations in vertical/nested manner, above or below another agg clause.

Given the initial aggregation:

```

A—> B
└─> C

```

If *insert_below* = 'A':

```

A—> new—> B
    └─> C

```

If *insert_above* = 'B':

```

A—> new—> B
└─> C

```

by argument accepts single occurrence or sequence of following formats:

- string (for terms agg concise declaration)
- regular Elasticsearch dict syntax
- AggNode instance (for instance Terms, Filters etc)

If *insert_below* nor *insert_above* is provided by will be placed between the the deepest linear bucket aggregation if there is no ambiguity, and its children:

```
A—> B      : OK generates      A—> B -> C -> by
A—> B      : KO, ambiguous, must precise either A, B or C
└─> C
```

Accepted all Aggs.__init__ syntaxes

```
>>> Aggs()\
>>> .groupby('terms', name='per_user_id', field='user_id')
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Passing a dict:

```
>>> Aggs().groupby({"terms_on_my_field":{"terms":{"field":"some_field"}}})
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Using DSL class:

```
>>> from pandagg.aggs import Terms
>>> Aggs().groupby(Terms('terms_on_my_field', field='some_field'))
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Shortcut syntax for terms aggregation: creates a terms aggregation, using field as aggregation name

```
>>> Aggs().groupby('some_field')
{"some_field":{"terms":{"field":"some_field"}}
```

Using a Aggs object:

```
>>> Aggs().groupby(Aggs('per_user_id', 'terms', field='user_id'))
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Accepted declarations for multiple aggregations:

Keyword Arguments

- *insert_below* (string) – Parent aggregation name under which these aggregations should be placed
- *insert_above* (string) – Aggregation name above which these aggregations should be placed
- *at_root* (string) – Insert aggregations at root of aggregation query
- remaining kwargs: Used as body in aggregation

Return type *pandagg.aggs.Aggs*

node_class

alias of *pandagg.node.aggs.abstract.AggNode*

show (*args, **kwargs)

Return tree structure in hierarchy style.

Parameters

- **nid** – Node identifier from which tree traversal will start. If None tree root will be used
- **filter_** – filter function performed on nodes. Nodes excluded from filter function nor their children won't be displayed
- **reverse** – the `reverse` param for sorting Node objects in the same level
- **key** – key used to order nodes of same parent
- **reverse** – reverse parameter applied at sorting
- **line_type** – display type choice
- **limit** – int, truncate tree display to this number of lines
- **kwargs** – kwargs params passed to node `line_repr` method

Return type unicode in python2, str in python3

to_dict (*from_=None, depth=None, with_name=True*)

pandagg.tree.aggs.bucket module

```
class pandagg.tree.aggs.bucket.Composite(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'composite'

class pandagg.tree.aggs.bucket.DateHistogram(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'date_histogram'

class pandagg.tree.aggs.bucket.DateRange(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'date_range'

class pandagg.tree.aggs.bucket.Filter(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'filter'

class pandagg.tree.aggs.bucket.Filters(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'filters'

class pandagg.tree.aggs.bucket.Global(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'global'

class pandagg.tree.aggs.bucket.Histogram(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'histogram'

class pandagg.tree.aggs.bucket.Missing(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg
    KEY = 'missing'
```

```
class pandagg.tree.aggs.bucket.Nested(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'nested'

class pandagg.tree.aggs.bucket.Range(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'range'

class pandagg.tree.aggs.bucket.ReverseNested(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'reverse_nested'

class pandagg.tree.aggs.bucket.Terms(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'terms'
```

pandagg.tree.aggs.metric module

```
class pandagg.tree.aggs.metric.Avg(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'avg'

class pandagg.tree.aggs.metric.Cardinality(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'cardinality'

class pandagg.tree.aggs.metric.ExtendedStats(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'extended_stats'

class pandagg.tree.aggs.metric.GeoBound(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'geo_bounds'

class pandagg.tree.aggs.metric.GeoCentroid(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'geo_centroid'

class pandagg.tree.aggs.metric.Max(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'max'

class pandagg.tree.aggs.metric.Min(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'min'

class pandagg.tree.aggs.metric.PercentileRanks(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'percentile_ranks'
```

```

class pandagg.tree.aggs.metric.Percentiles(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    Percents body argument can be passed to specify which percentiles to fetch.

    KEY = 'percentiles'

class pandagg.tree.aggs.metric.Stats(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'stats'

class pandagg.tree.aggs.metric.Sum(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'sum'

class pandagg.tree.aggs.metric.TopHits(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'top_hits'

class pandagg.tree.aggs.metric.ValueCount(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'value_count'

```

pandagg.tree.aggs.pipeline module

AbstractParentAgg aggregations: <https://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-pipeline.html>

```

class pandagg.tree.aggs.pipeline.AvgBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'avg_bucket'

class pandagg.tree.aggs.pipeline.BucketScript(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'bucket_script'

class pandagg.tree.aggs.pipeline.BucketSelector(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'bucket_selector'

class pandagg.tree.aggs.pipeline.BucketSort(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'bucket_sort'

class pandagg.tree.aggs.pipeline.CumulativeSum(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'cumulative_sum'

class pandagg.tree.aggs.pipeline.Derivative(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'derivative'

class pandagg.tree.aggs.pipeline.ExtendedStatsBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

```

```
KEY = 'extended_stats_bucket'

class pandagg.tree.aggs.pipeline.MaxBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'max_bucket'

class pandagg.tree.aggs.pipeline.MinBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'min_bucket'

class pandagg.tree.aggs.pipeline.MovingAvg(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'moving_avg'

class pandagg.tree.aggs.pipeline.PercentilesBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'percentiles_bucket'

class pandagg.tree.aggs.pipeline.SerialDiff(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'serial_diff'

class pandagg.tree.aggs.pipeline.StatsBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'stats_bucket'

class pandagg.tree.aggs.pipeline.SumBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'sum_bucket'
```

Module contents

pandagg.tree.query package

Submodules

pandagg.tree.query.abstract module

```
class pandagg.tree.query.abstract.Compound(**kwargs)
    Bases: pandagg.tree.query.abstract.Query

    KEY = None

class pandagg.tree.query.abstract.Leaf(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Query

    KEY = None

class pandagg.tree.query.abstract.Query(*args, **kwargs)
    Bases: pandagg.tree._tree.Tree
```

Combination of query clauses.

Mapping declaration is optional, but doing so validates query validity and automatically inserts nested clauses when necessary.

Keyword Arguments

- *mapping* (dict or `pandagg.tree.mapping.Mapping`) – Mapping of requested indice(s). Providing it will add validation features, and add required nested clauses if missing.
- *nested_autocorrect* (bool) – In case of missing nested clauses in query, if True, automatically add missing nested clauses, else raise error.
- remaining kwargs: Used as body in query clauses.

KEY = None

applied_nested_path_at_node (*nid*)

bool (*args, **kwargs)

boost (*args, **kwargs)

constant_score (*args, **kwargs)

dis_max (*args, **kwargs)

filter (*args, **kwargs)

function_score (*args, **kwargs)

has_child (*args, **kwargs)

has_parent (*args, **kwargs)

must (*args, **kwargs)

must_not (*args, **kwargs)

nested (*args, **kwargs)

node_class

alias of `pandagg.node.query.abstract.QueryClause`

parent_id (*args, **kwargs)

pinned_query (*args, **kwargs)

query (*args, **kwargs)

Insert new clause(s) in current query.

Inserted clause can accepts following syntaxes.

Given an empty query:

```
>>> from pandagg.query import Query
>>> q = Query()
```

flat syntax: clause type, followed by query clause body as keyword arguments:

```
>>> q.query('term', some_field=23)
{'term': {'some_field': 23}}
```

from regular Elasticsearch dict query:

```
>>> q.query({'term': {'some_field': 23}})
{'term': {'some_field': 23}}
```

using pandagg DSL:

```
>>> from pandagg.query import Term
>>> q.query(Term(field=23))
{'term': {'some_field': 23}}
```

Keyword Arguments

- *parent* (*str*) – named query clause under which the inserted clauses should be placed.
- *parent_param* (*str* optional parameter when using *parent* param) – parameter under which inserted clauses will be placed. For instance if *parent* clause is a boolean, can be ‘must’, ‘filter’, ‘should’, ‘must_not’.
- *child* (*str*) – named query clause above which the inserted clauses should be placed.
- *child_param* (*str* optional parameter when using *parent* param) – parameter of inserted boolean clause under which child clauses will be placed. For instance if inserted clause is a boolean, can be ‘must’, ‘filter’, ‘should’, ‘must_not’.
- *mode* (*str* one of ‘add’, ‘replace’, ‘replace_all’) – merging strategy when inserting clauses on a existing compound clause.
 - ‘add’ (default) : adds new clauses keeping initial ones
 - ‘replace’ : for each parameter (for instance in ‘bool’ case : ‘filter’, ‘must’, ‘must_not’, ‘should’), replace existing clauses under this parameter, by new ones only if declared in inserted compound query
 - ‘replace_all’ : existing compound clause is completely replaced by the new one

script_score (**args*, ***kwargs*)

should (**args*, ***kwargs*)

show (**args*, ***kwargs*)

Return tree structure in hierarchy style.

Parameters

- **nid** – Node identifier from which tree traversal will start. If None tree root will be used
- **filter_** – filter function performed on nodes. Nodes excluded from filter function nor their children won’t be displayed
- **reverse** – the *reverse* param for sorting Node objects in the same level
- **key** – key used to order nodes of same parent
- **reverse** – reverse parameter applied at sorting
- **line_type** – display type choice
- **limit** – int, truncate tree display to this number of lines
- **kwargs** – kwargs params passed to node *line_repr* method

Return type unicode in python2, str in python3

to_dict (*from_=None*, *with_name=True*)

Serialize query as native dict. :param *from_*: optional, :param *with_name*: optional :return:

pandagg.tree.query.compound module

```
class pandagg.tree.query.compound.Bool (**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'bool'

class pandagg.tree.query.compound.Boosting (**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'boosting'

class pandagg.tree.query.compound.ConstantScore (**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'constant_score'

class pandagg.tree.query.compound.DisMax (**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'dis_max'

class pandagg.tree.query.compound.FunctionScore (**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'function_score'
```

pandagg.tree.query.full_text module

```
class pandagg.tree.query.full_text.Common (*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'common'

class pandagg.tree.query.full_text.Intervals (*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'intervals'

class pandagg.tree.query.full_text.Match (*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match'

class pandagg.tree.query.full_text.MatchBoolPrefix (*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match_bool_prefix'

class pandagg.tree.query.full_text.MatchPhrase (*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match_phrase'

class pandagg.tree.query.full_text.MatchPhrasePrefix (*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match_phrase_prefix'

class pandagg.tree.query.full_text.MultiMatch (*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'multi_match'
```

```
class pandagg.tree.query.full_text.QueryString(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'query_string'

class pandagg.tree.query.full_text.SimpleQueryString(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'simple_string'
```

pandagg.tree.query.geo module

```
class pandagg.tree.query.geo.GeoBoundingBox(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_bounding_box'

class pandagg.tree.query.geo.GeoDistance(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_distance'

class pandagg.tree.query.geo.GeoPolygone(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_polygon'

class pandagg.tree.query.geo.GeoShape(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_shape'
```

pandagg.tree.query.joining module

```
class pandagg.tree.query.joining.HasChild(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'has_child'

class pandagg.tree.query.joining.HasParent(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'has_parent'

class pandagg.tree.query.joining.Nested(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'nested'

class pandagg.tree.query.joining.ParentId(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'parent_id'
```

pandagg.tree.query.shape module

```
class pandagg.tree.query.shape.Shape(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'shape'
```

pandagg.tree.query.span module

pandagg.tree.query.specialized module

```
class pandagg.tree.query.specialized.DistanceFeature(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'distance_feature'

class pandagg.tree.query.specialized.MoreLikeThis(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'more_like_this'

class pandagg.tree.query.specialized.Percolate(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'percolate'

class pandagg.tree.query.specialized.RankFeature(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'rank_feature'

class pandagg.tree.query.specialized.Script(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'script'

class pandagg.tree.query.specialized.Wrapper(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'wrapper'
```

pandagg.tree.query.specialized_compound module

```
class pandagg.tree.query.specialized_compound.PinnedQuery(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'pinned'

class pandagg.tree.query.specialized_compound.ScriptScore(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'script_score'
```

pandagg.tree.query.term_level module

```
class pandagg.tree.query.term_level.Exists(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'exists'

class pandagg.tree.query.term_level.Fuzzy(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'fuzzy'

class pandagg.tree.query.term_level.Ids(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf
```

```
KEY = 'ids'

class pandagg.tree.query.term_level.Prefix(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'prefix'

class pandagg.tree.query.term_level.Range(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'range'

class pandagg.tree.query.term_level.Regexp(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'regexp'

class pandagg.tree.query.term_level.Term(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'term'

class pandagg.tree.query.term_level.Terms(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'terms'

class pandagg.tree.query.term_level.TermsSet(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'terms_set'

class pandagg.tree.query.term_level.Type(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'type'

class pandagg.tree.query.term_level.Wildcard(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'wildcard'
```

Module contents

5.1.3.2 Submodules

pandagg.tree.mapping module

```
class pandagg.tree.mapping.Mapping(*args, **kwargs)
    Bases: pandagg.tree._tree.Tree

    KEY = None

    get(key)
        Get a node by its id. :param nid: str, identifier of node to fetch :rtype: lighttree.node.Node

    list_nesteds_at_field(field_path)

    mapping_type_of_field(field_path)

    nested_at_field(field_path)

    node_class
        alias of pandagg.node.mapping.abstract.Field
```

node_path (*nid*)

resolve_path_to_id (*path*)

to_dict (*from_=None, depth=None*)

validate_agg_node (*agg_node, exc=True*)

Ensure if node has field or path that it exists in mapping, and that required aggregation type if allowed on this kind of field. :param agg_node: AggNode you want to validate on this mapping :param exc: boolean, if set to True raise exception if invalid :rtype: boolean

pandagg.tree.response module

class pandagg.tree.response.**AggsResponseTree** (*aggs, index*)

Bases: pandagg.tree._tree.Tree

Tree representation of an ElasticSearch response.

bucket_properties (*bucket, properties=None, end_level=None, depth=None*)

Recursive method returning a given bucket's properties in the form of an ordered dictionary. Travel from current bucket through all ancestors until reaching root.

Parameters

- **bucket** – instance of pandagg.buckets.buckets.Bucket
- **properties** – OrderedDict accumulator of 'level' -> 'key'
- **end_level** – optional parameter to specify until which level properties are fetched
- **depth** – optional parameter to specify a limit number of levels which are fetched

Returns OrderedDict of structure 'level' -> 'key'

get_bucket_filter (*nid*)

Build query filtering documents belonging to that bucket. Suppose the following configuration:

```
Base          <- filter on base
|— Nested_A   no filter on A (nested still must be applied)
↪for children
|   |— SubNested A1
|   |— SubNested A2   <- filter on A2
|— Nested_B          <- filter on B
```

parse (*raw_response*)

Build response tree from ElasticSearch aggregation response

Note: if the root aggregation node can generate multiple buckets, a response root is crafted to avoid having multiple roots.

Parameters **raw_response** – ElasticSearch aggregation response

Returns self

show (***kwargs*)

Return tree structure in hierarchy style.

Parameters

- **nid** – Node identifier from which tree traversal will start. If None tree root will be used
- **filter_** – filter function performed on nodes. Nodes excluded from filter function nor their children won't be displayed

- **reverse** – the `reverse` param for sorting `Node` objects in the same level
- **key** – key used to order nodes of same parent
- **reverse** – reverse parameter applied at sorting
- **line_type** – display type choice
- **limit** – int, truncate tree display to this number of lines
- **kwargs** – kwargs params passed to node `line_repr` method

Return type unicode in python2, str in python3

5.1.3.3 Module contents

5.2 Submodules

5.2.1 pandagg.aggs module

class pandagg.aggs.**Aggs** (*args, **kwargs)

Bases: pandagg.tree._tree.Tree

Combination of aggregation clauses. This class provides handful methods to build an aggregation (see `aggs()` and `groupby()`), and is used as well to parse aggregations response in handy formats.

Mapping declaration is optional, but doing so validates aggregation validity and automatically handles missing nested clauses.

All following syntaxes are identical:

From a dict:

```
>>> Aggs({"per_user":{"terms":{"field":"user"}}})
```

Using shortcut declaration: first argument is the aggregation type, other arguments are aggregation body parameters:

```
>>> Aggs('terms', name='per_user', field='user')
```

Using DSL class:

```
>>> from pandagg.aggs import Terms
>>> Aggs(Terms('per_user', field='user'))
```

Dict and DSL class syntaxes allow to provide multiple clauses aggregations:

```
>>> Aggs({"per_user":{"terms":{"field":"user"}, "aggs": {"avg_age": {"avg": {
↪ "field": "age"}}}}})
```

Which is similar to:

```
>>> from pandagg.aggs import Terms, Avg
>>> Terms('per_user', field='user', aggs=Avg('avg_age', field='age'))
```

Keyword Arguments

- *mapping* (dict or `pandagg.tree.mapping.Mapping`) – Mapping of requested indices. Providing it will validate aggregations validity, and add required nested clauses if missing.
- *nested_autocorrect* (bool) – In case of missing nested clauses in aggregation, if True, automatically add missing nested clauses, else raise error.
- remaining kwargs: Used as body in aggregation

aggs (*args, **kwargs)

Arrange passed aggregations “horizontally”.

Given the initial aggregation:

```
A—> B
└─> C
```

If passing multiple aggregations with *insert_below* = ‘A’:

```
A—> B
└─> C
  └─> new1
    └─> new2
```

Note: those will be placed under the *insert_below* aggregation clause id if provided, else under the deepest linear bucket aggregation if there is no ambiguity:

OK:

```
A—> B -> C -> new
```

KO:

```
A—> B
└─> C
```

args accepts single occurrence or sequence of following formats:

- string (for terms agg concise declaration)
- regular Elasticsearch dict syntax
- AggNode instance (for instance Terms, Filters etc)

Keyword Arguments

- *insert_below* (string) – Parent aggregation name under which these aggregations should be placed
- *at_root* (string) – Insert aggregations at root of aggregation query
- remaining kwargs: Used as body in aggregation

Return type `pandagg.aggs.Aggs`

applied_nested_path_at_node (*nid*)

deepest_linear_bucket_agg

Return deepest bucket aggregation node (`pandagg.nodes.abstract.BucketAggNode`) of that aggregation that neither has siblings, nor has an ancestor with siblings.

groupby (*args, **kwargs)

Arrange passed aggregations in vertical/nested manner, above or below another agg clause.

Given the initial aggregation:

```
A—> B
└─> C
```

If *insert_below* = 'A':

```
A—> new—> B
      └─> C
```

If *insert_above* = 'B':

```
A—> new—> B
└─> C
```

by argument accepts single occurrence or sequence of following formats:

- string (for terms agg concise declaration)
- regular Elasticsearch dict syntax
- AggNode instance (for instance Terms, Filters etc)

If *insert_below* nor *insert_above* is provided by will be placed between the the deepest linear bucket aggregation if there is no ambiguity, and its children:

```
A—> B      : OK generates      A—> B -> C -> by
A—> B      : KO, ambiguous, must precise either A, B or C
└─> C
```

Accepted all Aggs.__init__ syntaxes

```
>>> Aggs() \
>>> .groupby('terms', name='per_user_id', field='user_id')
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Passing a dict:

```
>>> Aggs().groupby({"terms_on_my_field":{"terms":{"field":"some_field"}}})
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Using DSL class:

```
>>> from pandagg.aggs import Terms
>>> Aggs().groupby(Terms('terms_on_my_field', field='some_field'))
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Shortcut syntax for terms aggregation: creates a terms aggregation, using field as aggregation name

```
>>> Aggs().groupby('some_field')
{"some_field":{"terms":{"field":"some_field"}}
```

Using a Aggs object:

```
>>> Aggs().groupby(Aggs('per_user_id', 'terms', field='user_id'))
{"terms_on_my_field":{"terms":{"field":"some_field"}}
```

Accepted declarations for multiple aggregations:

Keyword Arguments

- *insert_below* (string) – Parent aggregation name under which these aggregations should be placed
- *insert_above* (string) – Aggregation name above which these aggregations should be placed
- *at_root* (string) – Insert aggregations at root of aggregation query
- remaining kwargs: Used as body in aggregation

Return type *pandagg.aggs.Aggs*

node_class

alias of *pandagg.node.aggs.abstract.AggsNode*

show (*args, **kwargs)

Return tree structure in hierarchy style.

Parameters

- **nid** – Node identifier from which tree traversal will start. If None tree root will be used
- **filter_** – filter function performed on nodes. Nodes excluded from filter function nor their children won't be displayed
- **reverse** – the *reverse* param for sorting Node objects in the same level
- **key** – key used to order nodes of same parent
- **reverse** – reverse parameter applied at sorting
- **line_type** – display type choice
- **limit** – int, truncate tree display to this number of lines
- **kwargs** – kwargs params passed to node *line_repr* method

Return type unicode in python2, str in python3

to_dict (from_=None, depth=None, with_name=True)

class *pandagg.aggs.Terms* (*args, **kwargs)

Bases: *pandagg.tree.aggs.aggs.AbstractParentAgg*

KEY = 'terms'

class *pandagg.aggs.Filters* (*args, **kwargs)

Bases: *pandagg.tree.aggs.aggs.AbstractParentAgg*

KEY = 'filters'

class *pandagg.aggs.Histogram* (*args, **kwargs)

Bases: *pandagg.tree.aggs.aggs.AbstractParentAgg*

KEY = 'histogram'

class *pandagg.aggs.DateHistogram* (*args, **kwargs)

Bases: *pandagg.tree.aggs.aggs.AbstractParentAgg*

KEY = 'date_histogram'

class *pandagg.aggs.Range* (*args, **kwargs)

Bases: *pandagg.tree.aggs.aggs.AbstractParentAgg*

```
    KEY = 'range'

class pandagg.aggs.Global(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'global'

class pandagg.aggs.Filter(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'filter'

class pandagg.aggs.Missing(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'missing'

class pandagg.aggs.Nested(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'nested'

class pandagg.aggs.ReverseNested(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'reverse_nested'

class pandagg.aggs.Avg(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'avg'

class pandagg.aggs.Max(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'max'

class pandagg.aggs.Sum(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'sum'

class pandagg.aggs.Min(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'min'

class pandagg.aggs.Cardinality(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'cardinality'

class pandagg.aggs.Stats(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'stats'

class pandagg.aggs.ExtendedStats(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'extended_stats'

class pandagg.aggs.Percentiles(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    Percents body argument can be passed to specify which percentiles to fetch.
```

```
    KEY = 'percentiles'

class pandagg.aggs.PercentileRanks(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'percentile_ranks'

class pandagg.aggs.GeoBound(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'geo_bounds'

class pandagg.aggs.GeoCentroid(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'geo_centroid'

class pandagg.aggs.TopHits(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'top_hits'

class pandagg.aggs.ValueCount(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractLeafAgg

    KEY = 'value_count'

class pandagg.aggs.AvgBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'avg_bucket'

class pandagg.aggs.Derivative(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'derivative'

class pandagg.aggs.MaxBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'max_bucket'

class pandagg.aggs.MinBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'min_bucket'

class pandagg.aggs.SumBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'sum_bucket'

class pandagg.aggs.StatsBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'stats_bucket'

class pandagg.aggs.ExtendedStatsBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'extended_stats_bucket'

class pandagg.aggs.PercentilesBucket(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'percentiles_bucket'
```

```
class pandagg.aggs.MovingAvg(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'moving_avg'

class pandagg.aggs.CumulativeSum(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'cumulative_sum'

class pandagg.aggs.BucketScript(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'bucket_script'

class pandagg.aggs.BucketSelector(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'bucket_selector'

class pandagg.aggs.BucketSort(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'bucket_sort'

class pandagg.aggs.SerialDiff(*args, **kwargs)
    Bases: pandagg.tree.aggs.aggs.AbstractParentAgg

    KEY = 'serial_diff'
```

5.2.2 pandagg.connections module

```
class pandagg.connections.Connections
    Bases: object
```

Class responsible for holding connections to different clusters. Used as a singleton in this module.

add_connection (*alias*, *conn*)

Add a connection object, it will be passed through as-is.

configure (***kwargs*)

Configure multiple connections at once, useful for passing in config dictionaries obtained from other sources, like Django's settings or a configuration management tool.

Example:

```
connections.configure(
    default={'hosts': 'localhost'},
    dev={'hosts': ['esdev1.example.com:9200'], 'sniff_on_start': True},
)
```

Connections will only be constructed lazily when requested through `get_connection`.

create_connection (*alias*='default', ***kwargs*)

Construct an instance of `elasticsearch.Elasticsearch` and register it under given alias.

get_connection (*alias*='default')

Retrieve a connection, construct it if necessary (only configuration was passed to us). If a non-string alias has been passed through we assume it's already a client instance and will just return it as-is.

Raises `KeyError` if no client (or its definition) is registered under the alias.

remove_connection (*alias*)

Remove connection from the registry. Raises `KeyError` if connection wasn't found.

5.2.3 pandagg.discovery module

class pandagg.discovery.**Index** (*name, settings, mapping, aliases, client=None*)

Bases: `object`

search (*nested_autocorrect=True*)

class pandagg.discovery.**Indices** (***kwargs*)

Bases: `lighttree.interactive.Obj`

pandagg.discovery.**discover** (*using, index='*'*)

Parameters

- **using** – Elasticsearch client
- **index** – Comma-separated list or wildcard expression of index names used to limit the request.

5.2.4 pandagg.exceptions module

exception pandagg.exceptions.**AbsentMappingFieldError**

Bases: `pandagg.exceptions.MappingError`

Field is not present in mapping.

exception pandagg.exceptions.**InvalidAggregation**

Bases: `Exception`

Wrong aggregation definition

exception pandagg.exceptions.**InvalidOperationMappingFieldError**

Bases: `pandagg.exceptions.MappingError`

Invalid aggregation type on this mapping field.

exception pandagg.exceptions.**MappingError**

Bases: `Exception`

Basic Mapping Error

exception pandagg.exceptions.**VersionIncompatibilityError**

Bases: `Exception`

Pandagg is not compatible with this ElasticSearch version.

5.2.5 pandagg.mapping module

class pandagg.mapping.**Mapping** (**args, **kwargs*)

Bases: `pandagg.tree._tree.Tree`

KEY = `None`

get (*key*)

Get a node by its id. :param nid: str, identifier of node to fetch :rtype: `lighttree.node.Node`

list_nesteds_at_field (*field_path*)

```
mapping_type_of_field(field_path)
nested_at_field(field_path)
node_class
    alias of pandagg.node.mapping.abstract.Field
node_path(nid)
resolve_path_to_id(path)
to_dict(from_=None, depth=None)
validate_agg_node(agg_node, exc=True)
    Ensure if node has field or path that it exists in mapping, and that required aggregation type if allowed on
    this kind of field. :param agg_node: AggNode you want to validate on this mapping :param exc: boolean,
    if set to True raise exception if invalid :rtype: boolean

class pandagg.mapping.IMapping(*args, **kwargs)
    Bases: lighttree.interactive.TreeBasedObj

    Interactive wrapper upon mapping tree, allowing field navigation and quick access to single clause aggregations
    computation.

class pandagg.mapping.Text(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'text'

class pandagg.mapping.Keyword(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'keyword'

class pandagg.mapping.Long(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'long'

class pandagg.mapping.Integer(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'integer'

class pandagg.mapping.Short(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'short'

class pandagg.mapping.Byte(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'byte'

class pandagg.mapping.Double(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'double'

class pandagg.mapping.HalfFloat(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'half_float'

class pandagg.mapping.ScaledFloat(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField
```



```
    KEY = 'scaled_float'

class pandagg.mapping.Date(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'date'

class pandagg.mapping.DateNanos(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'date_nanos'

class pandagg.mapping.Boolean(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'boolean'

class pandagg.mapping.Binary(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'binary'

class pandagg.mapping.IntegerRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'integer_range'

class pandagg.mapping.Float(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'float'

class pandagg.mapping.FloatRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'float_range'

class pandagg.mapping.LongRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'long_range'

class pandagg.mapping.DoubleRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'double_range'

class pandagg.mapping.DateRange(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    KEY = 'date_range'

class pandagg.mapping.Object(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedComplexField

    KEY = 'object'

class pandagg.mapping.Nested(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedComplexField

    KEY = 'nested'

class pandagg.mapping.GeoPoint(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedRegularField

    For lat/lon points
```

```
KEY = 'geo_point'
```

class pandagg.mapping.GeoShape (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
For complex shapes like polygons

```
KEY = 'geo_shape'
```

class pandagg.mapping.IP (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
for IPv4 and IPv6 addresses

```
KEY = 'IP'
```

class pandagg.mapping.Completion (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
To provide auto-complete suggestions

```
KEY = 'completion'
```

class pandagg.mapping.TokenCount (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
To count the number of tokens in a string

```
KEY = 'token_count'
```

class pandagg.mapping.MapperMurMur3 (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
To compute hashes of values at index-time and store them in the index

```
KEY = 'murmur3'
```

class pandagg.mapping.MapperAnnotatedText (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
To index text containing special markup (typically used for identifying named entities)

```
KEY = 'annotated-text'
```

class pandagg.mapping.Percolator (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
Accepts queries from the query-dsl

```
KEY = 'percolator'
```

class pandagg.mapping.Join (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
Defines parent/child relation for documents within the same index

```
KEY = 'join'
```

class pandagg.mapping.RankFeature (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*
Record numeric feature to boost hits at query time.

```
KEY = 'rank_feature'
```

class pandagg.mapping.RankFeatures (**body)
Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

Record numeric features to boost hits at query time.

KEY = 'rank_features'

class pandagg.mapping.**DenseVector**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

Record dense vectors of float values.

KEY = 'dense_vector'

class pandagg.mapping.**SparseVector**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

Record sparse vectors of float values.

KEY = 'sparse_vector'

class pandagg.mapping.**SearchAsYouType**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

A text-like field optimized for queries to implement as-you-type completion

KEY = 'search_as_you_type'

class pandagg.mapping.**Alias**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

Defines an alias to an existing field.

KEY = 'alias'

class pandagg.mapping.**Flattened**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

Allows an entire JSON object to be indexed as a single field.

KEY = 'flattened'

class pandagg.mapping.**Shape**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

For arbitrary cartesian geometries.

KEY = 'shape'

class pandagg.mapping.**Histogram**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedRegularField*

For pre-aggregated numerical values for percentiles aggregations.

KEY = 'histogram'

class pandagg.mapping.**Index**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedField*

The index to which the document belongs.

KEY = '_index'

class pandagg.mapping.**Type**(**body)

Bases: *pandagg.node.mapping.abstract.UnnamedField*

The document's mapping type.

KEY = '_type'

```
class pandagg.mapping.Id(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField
    The document's ID.
    KEY = '_id'

class pandagg.mapping.FieldNames(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField
    All fields in the document which contain non-null values.
    KEY = '_field_names'

class pandagg.mapping.Source(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField
    The original JSON representing the body of the document.
    KEY = '_source'

class pandagg.mapping.Size(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField
    The size of the _source field in bytes, provided by the mapper-size plugin.
    KEY = '_size'

class pandagg.mapping.Ignored(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField
    All fields in the document that have been ignored at index time because of ignore_malformed.
    KEY = '_ignored'

class pandagg.mapping.Routing(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField
    A custom routing value which routes a document to a particular shard.
    KEY = '_routing'

class pandagg.mapping.Meta(**body)
    Bases: pandagg.node.mapping.abstract.UnnamedField
    Application specific metadata.
    KEY = '_meta'
```

5.2.6 pandagg.query module

```
class pandagg.query.Query(*args, **kwargs)
    Bases: pandagg.tree._tree.Tree
    Combination of query clauses.
    Mapping declaration is optional, but doing so validates query validity and automatically inserts nested clauses when necessary.
```

Keyword Arguments

- *mapping* (dict or *pandagg.tree.mapping.Mapping*) – Mapping of requested indice(s). Providing it will add validation features, and add required nested clauses if missing.
- *nested_autocorrect* (bool) – In case of missing nested clauses in query, if True, automatically add missing nested clauses, else raise error.

- remaining kwargs: Used as body in query clauses.

```

KEY = None
applied_nested_path_at_node (nid)
bool (*args, **kwargs)
boost (*args, **kwargs)
constant_score (*args, **kwargs)
dis_max (*args, **kwargs)
filter (*args, **kwargs)
function_score (*args, **kwargs)
has_child (*args, **kwargs)
has_parent (*args, **kwargs)
must (*args, **kwargs)
must_not (*args, **kwargs)
nested (*args, **kwargs)
node_class
    alias of pandagg.node.query.abstract.QueryClause
parent_id (*args, **kwargs)
pinned_query (*args, **kwargs)
query (*args, **kwargs)
    Insert new clause(s) in current query.

```

Inserted clause can accepts following syntaxes.

Given an empty query:

```

>>> from pandagg.query import Query
>>> q = Query()

```

flat syntax: clause type, followed by query clause body as keyword arguments:

```

>>> q.query('term', some_field=23)
{'term': {'some_field': 23}}

```

from regular Elasticsearch dict query:

```

>>> q.query({'term': {'some_field': 23}})
{'term': {'some_field': 23}}

```

using pandagg DSL:

```

>>> from pandagg.query import Term
>>> q.query(Term(field=23))
{'term': {'some_field': 23}}

```

Keyword Arguments

- *parent* (*str*) – named query clause under which the inserted clauses should be placed.

- *parent_param* (str optional parameter when using *parent* param) – parameter under which inserted clauses will be placed. For instance if *parent* clause is a boolean, can be ‘must’, ‘filter’, ‘should’, ‘must_not’.
- *child* (str) – named query clause above which the inserted clauses should be placed.
- *child_param* (str optional parameter when using *parent* param) – parameter of inserted boolean clause under which child clauses will be placed. For instance if inserted clause is a boolean, can be ‘must’, ‘filter’, ‘should’, ‘must_not’.
- *mode* (str one of ‘add’, ‘replace’, ‘replace_all’) – merging strategy when inserting clauses on a existing compound clause.
 - ‘add’ (default) : adds new clauses keeping initial ones
 - ‘replace’ : for each parameter (for instance in ‘bool’ case : ‘filter’, ‘must’, ‘must_not’, ‘should’), replace existing clauses under this parameter, by new ones only if declared in inserted compound query
 - ‘replace_all’ : existing compound clause is completely replaced by the new one

script_score (*args, **kwargs)

should (*args, **kwargs)

show (*args, **kwargs)

Return tree structure in hierarchy style.

Parameters

- **nid** – Node identifier from which tree traversal will start. If None tree root will be used
- **filter_** – filter function performed on nodes. Nodes excluded from filter function nor their children won’t be displayed
- **reverse** – the *reverse* param for sorting *Node* objects in the same level
- **key** – key used to order nodes of same parent
- **reverse** – reverse parameter applied at sorting
- **line_type** – display type choice
- **limit** – int, truncate tree display to this number of lines
- **kwargs** – kwargs params passed to node *line_repr* method

Return type unicode in python2, str in python3

to_dict (from_=None, with_name=True)

Serialize query as native dict. :param from_: optional, :param with_name: optional :return:

class pandagg.query.Exists (*args, **kwargs)

Bases: *pandagg.tree.query.abstract.Leaf*

KEY = ‘exists’

class pandagg.query.Fuzzy (*args, **kwargs)

Bases: *pandagg.tree.query.abstract.Leaf*

KEY = ‘fuzzy’

class pandagg.query.Ids (*args, **kwargs)

Bases: *pandagg.tree.query.abstract.Leaf*

KEY = ‘ids’

```

class pandagg.query.Prefix(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'prefix'

class pandagg.query.Range(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'range'

class pandagg.query.Regexp(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'regexp'

class pandagg.query.Term(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'term'

class pandagg.query.Terms(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'terms'

class pandagg.query.TermsSet(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'terms_set'

class pandagg.query.Type(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'type'

class pandagg.query.Wildcard(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'wildcard'

class pandagg.query.Intervals(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'intervals'

class pandagg.query.Match(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match'

class pandagg.query.MatchBoolPrefix(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match_bool_prefix'

class pandagg.query.MatchPhrase(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match_phrase'

class pandagg.query.MatchPhrasePrefix(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'match_phrase_prefix'

class pandagg.query.MultiMatch(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

```

```
KEY = 'multi_match'

class pandagg.query.Common(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'common'

class pandagg.query.QueryString(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'query_string'

class pandagg.query.SimpleQueryString(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'simple_string'

class pandagg.query.Bool(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'bool'

class pandagg.query.Boosting(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'boosting'

class pandagg.query.ConstantScore(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'constant_score'

class pandagg.query.FunctionScore(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'function_score'

class pandagg.query.DisMax(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'dis_max'

class pandagg.query.Nested(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'nested'

class pandagg.query.HasParent(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'has_parent'

class pandagg.query.HasChild(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'has_child'

class pandagg.query.ParentId(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'parent_id'

class pandagg.query.Shape(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'shape'
```



```
class pandagg.query.GeoShape(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_shape'

class pandagg.query.GeoPolygone(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_polygon'

class pandagg.query.GeoDistance(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_distance'

class pandagg.query.GeoBoundingBox(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'geo_bounding_box'

class pandagg.query.DistanceFeature(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'distance_feature'

class pandagg.query.MoreLikeThis(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'more_like_this'

class pandagg.query.Percolate(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'percolate'

class pandagg.query.RankFeature(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'rank_feature'

class pandagg.query.Script(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'script'

class pandagg.query.Wrapper(*args, **kwargs)
    Bases: pandagg.tree.query.abstract.Leaf

    KEY = 'wrapper'

class pandagg.query.ScriptScore(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'script_score'

class pandagg.query.PinnedQuery(**kwargs)
    Bases: pandagg.tree.query.abstract.Compound

    KEY = 'pinned'
```

5.2.7 pandagg.response module

```
class pandagg.response.Aggregations(data, aggs, query, index, client)
    Bases: object
```

get (*key*)

keys ()

serialize (*output='tabular', **kwargs*)

Parameters

- **output** – output format, one of “raw”, “tree”, “interactive_tree”, “normalized”, “tabular”, “dataframe”
- **kwargs** – tabular serialization kwargs

Returns

to_dataframe (*grouped_by=None, normalize_children=True, with_single_bucket_groups=False*)

to_interactive_tree ()

to_normalized ()

to_tabular (*index_orient=True, grouped_by=None, expand_columns=True, expand_sep='|', normalize=True, with_single_bucket_groups=False*)

Build tabular view of ES response grouping levels (rows) until ‘grouped_by’ aggregation node included is reached, and using children aggregations of grouping level as values for each of generated groups (columns).

Suppose an aggregation of this shape (A & B bucket aggregations):

```
A—> B—> C1
      |> C2
      |> C3
```

With `grouped_by='B'`, breakdown ElasticSearch response (tree structure), into a tabular structure of this shape:

		C1	C2	C3
A	B			
wood	blue	10	4	0
	red	7	5	2
steel	blue	1	9	0
	red	23	4	2

Parameters

- **index_orient** – if True, level-key samples are returned as tuples, else in a dictionary
- **grouped_by** – name of the aggregation node used as last grouping level
- **normalize** – if True, normalize columns buckets

Returns index, index_names, values

to_tree ()

class pandagg.response.**Hit** (*data*)

Bases: `object`

class pandagg.response.**Hits** (*hits*)

Bases: `object`

class pandagg.response.**Response** (*data, search*)

Bases: `object`

success

5.2.8 pandagg.search module

class pandagg.search.**MultiSearch** (***kwargs*)

Bases: *pandagg.search.Request*

Combine multiple Search objects into a single request.

add (*search*)

Adds a new Search object to the request:

```
ms = MultiSearch(index='my-index')
ms = ms.add(Search(doc_type=Category).filter('term', category='python'))
ms = ms.add(Search(doc_type=Blog))
```

execute ()

Execute the multi search request and return a list of search results.

to_dict ()

class pandagg.search.**Request** (*using, index=None*)

Bases: *object*

index (**index*)

Set the index for the search. If called empty it will remove all information.

Example:

```
s = Search() s = s.index('twitter-2015.01.01', 'twitter-2015.01.02') s = s.index(['twitter-2015.01.01', 'twitter-2015.01.02'])
```

params (***kwargs*)

Specify query params to be used when executing the search. All the keyword arguments will override the current values. See <https://elasticsearch-py.readthedocs.io/en/master/api.html#elasticsearch.Elasticsearch.search> for all available parameters.

Example:

```
s = Search()
s = s.params(routing='user-1', preference='local')
```

using (*client*)

Associate the search request with an elasticsearch client. A fresh copy will be returned with current instance remaining unchanged.

Parameters **client** – an instance of `elasticsearch.Elasticsearch` to use or an alias to look up in `elasticsearch_dsl.connections`

class pandagg.search.**Search** (*using=None, index=None, mapping=None, nested_autocorrect=False*)

Bases: *pandagg.search.Request*

aggs (**args, **kwargs*)

Arrange passed aggregations “horizontally”.

Given the initial aggregation:

```
A → B
└→ C
```

If passing multiple aggregations with *insert_below* = 'A':

```
A—> B
└─> C
└─> new1
└─> new2
```

Note: those will be placed under the *insert_below* aggregation clause id if provided, else under the deepest linear bucket aggregation if there is no ambiguity:

OK:

```
A—> B -> C -> new
```

KO:

```
A—> B
└─> C
```

args accepts single occurrence or sequence of following formats:

- string (for terms agg concise declaration)
- regular Elasticsearch dict syntax
- AggNode instance (for instance Terms, Filters etc)

Keyword Arguments

- *insert_below* (string) – Parent aggregation name under which these aggregations should be placed
- *at_root* (string) – Insert aggregations at root of aggregation query
- remaining kwargs: Used as body in aggregation

Return type *pandagg.aggs.Aggs*

bool (*args, **kwargs)

count ()

Return the number of hits matching the query and filters. Note that only the actual number is returned.

delete () *executes the query by delegating to delete_by_query()*

exclude (*args, **kwargs)

Must not wrapped in filter context.

execute ()

Execute the search and return an instance of *Response* wrapping all the data.

filter (*args, **kwargs)

classmethod from_dict (d)

Construct a new *Search* instance from a raw dict containing the search body. Useful when migrating from raw dictionaries.

Example:

```
s = Search.from_dict({
    "query": {
        "bool": {
```

(continues on next page)

(continued from previous page)

```

        "must": [...]
    },
    "aggs": {...}
})
s = s.filter('term', published=True)

```

groupby (*args, **kwargs)

Arrange passed aggregations in vertical/nested manner, above or below another agg clause.

Given the initial aggregation:

```

A—> B
└─> C

```

If *insert_below* = 'A':

```

A—> new—> B
    └─> C

```

If *insert_above* = 'B':

```

A—> new—> B
└─> C

```

by argument accepts single occurrence or sequence of following formats:

- string (for terms agg concise declaration)
- regular Elasticsearch dict syntax
- AggNode instance (for instance Terms, Filters etc)

If *insert_below* nor *insert_above* is provided by will be placed between the the deepest linear bucket aggregation if there is no ambiguity, and its children:

```

A—> B      : OK generates      A—> B -> C -> by
A—> B      : KO, ambiguous, must precise either A, B or C
└─> C

```

Accepted all Aggs.__init__ syntaxes

```

>>> Aggs()\
>>> .groupby('terms', name='per_user_id', field='user_id')
{"terms_on_my_field":{"terms":{"field":"some_field"}}}

```

Passing a dict:

```

>>> Aggs().groupby({"terms_on_my_field":{"terms":{"field":"some_field"}}})
{"terms_on_my_field":{"terms":{"field":"some_field"}}}

```

Using DSL class:

```

>>> from pandagg.aggs import Terms
>>> Aggs().groupby(Terms('terms_on_my_field', field='some_field'))
{"terms_on_my_field":{"terms":{"field":"some_field"}}}

```

Shortcut syntax for terms aggregation: creates a terms aggregation, using field as aggregation name

```
>>> Aggs().groupby('some_field')
{"some_field":{"terms":{"field":"some_field"}}}
```

Using a Aggs object:

```
>>> Aggs().groupby(Aggs('per_user_id', 'terms', field='user_id'))
{"terms_on_my_field":{"terms":{"field":"some_field"}}}
```

Accepted declarations for multiple aggregations:

Keyword Arguments

- *insert_below* (string) – Parent aggregation name under which these aggregations should be placed
- *insert_above* (string) – Aggregation name above which these aggregations should be placed
- *at_root* (string) – Insert aggregations at root of aggregation query
- remaining kwargs: Used as body in aggregation

Return type *pandagg.aggs.Aggs*

highlight (*fields, **kwargs)

Request highlighting of some fields. All keyword arguments passed in will be used as parameters for all the fields in the *fields* parameter. Example:

```
Search().highlight('title', 'body', fragment_size=50)
```

will produce the equivalent of:

```
{
  "highlight": {
    "fields": {
      "body": {"fragment_size": 50},
      "title": {"fragment_size": 50}
    }
  }
}
```

If you want to have different options for different fields you can call *highlight* twice:

```
Search().highlight('title', fragment_size=50).highlight('body', fragment_
↪size=100)
```

which will produce:

```
{
  "highlight": {
    "fields": {
      "body": {"fragment_size": 100},
      "title": {"fragment_size": 50}
    }
  }
}
```

highlight_options (**kwargs)

Update the global highlighting options used for this request. For example:

```
s = Search()
s = s.highlight_options(order='score')
```

must (*args, **kwargs)

must_not (*args, **kwargs)

post_filter (*args, **kwargs)

query (*args, **kwargs)

Insert new clause(s) in current query.

Inserted clause can accepts following syntaxes.

Given an empty query:

```
>>> from pandagg.query import Query
>>> q = Query()
```

flat syntax: clause type, followed by query clause body as keyword arguments:

```
>>> q.query('term', some_field=23)
{'term': {'some_field': 23}}
```

from regular Elasticsearch dict query:

```
>>> q.query({'term': {'some_field': 23}})
{'term': {'some_field': 23}}
```

using pandagg DSL:

```
>>> from pandagg.query import Term
>>> q.query(Term(field=23))
{'term': {'some_field': 23}}
```

Keyword Arguments

- *parent* (str) – named query clause under which the inserted clauses should be placed.
- *parent_param* (str optional parameter when using *parent* param) – parameter under which inserted clauses will be placed. For instance if *parent* clause is a boolean, can be ‘must’, ‘filter’, ‘should’, ‘must_not’.
- *child* (str) – named query clause above which the inserted clauses should be placed.
- *child_param* (str optional parameter when using *parent* param) – parameter of inserted boolean clause under which child clauses will be placed. For instance if inserted clause is a boolean, can be ‘must’, ‘filter’, ‘should’, ‘must_not’.
- *mode* (str one of ‘add’, ‘replace’, ‘replace_all’) – merging strategy when inserting clauses on a existing compound clause.
 - ‘add’ (default) : adds new clauses keeping initial ones
 - ‘replace’ : for each parameter (for instance in ‘bool’ case : ‘filter’, ‘must’, ‘must_not’, ‘should’), replace existing clauses under this parameter, by new ones only if declared in inserted compound query
 - ‘replace_all’ : existing compound clause is completely replaced by the new one

scan()

Turn the search into a scan search and return a generator that will iterate over all the documents matching the query.

Use `params` method to specify any additional arguments you wish to pass to the underlying scan helper from `elasticsearch-py` - <https://elasticsearch-py.readthedocs.io/en/master/helpers.html#elasticsearch.helpers.scan>

script_fields(kwargs)**

Define script fields to be calculated on hits. See <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-script-fields.html> for more details.

Example:

```
s = Search()
s = s.script_fields(times_two="doc['field'].value * 2")
s = s.script_fields(
    times_three={
        'script': {
            'inline': "doc['field'].value * params.n",
            'params': {'n': 3}
        }
    }
)
```

should(*args, **kwargs)**size(size)**

Equivalent to:

```
s = Search().params(size=size)
```

sort(*keys)

Add sorting information to the search request. If called without arguments it will remove all sort requirements. Otherwise it will replace them. Acceptable arguments are:

```
'some.field'
'-some.other.field'
{'different.field': {'any': 'dict'}}
```

so for example:

```
s = Search().sort(
    'category',
    '-title',
    {"price": {"order": "asc", "mode": "avg"}}
)
```

will sort by category, title (in descending order) and price in ascending order using the avg mode.

The API returns a copy of the Search object and can thus be chained.

source(fields=None, **kwargs)

Selectively control how the `_source` field is returned.

Parameters **fields** – wildcard string, array of wildcards, or dictionary of includes and excludes

If `fields` is `None`, the entire document will be returned for each hit. If `fields` is a dictionary with keys of 'includes' and/or 'excludes' the fields will be either included or excluded appropriately.

Calling this multiple times with the same named parameter will override the previous values with the new ones.

Example:

```
s = Search()
s = s.source(includes=['obj1.*'], excludes=["*.description"])

s = Search()
s = s.source(includes=['obj1.*']).source(excludes=["*.description"])
```

suggest (*name*, *text*, ***kwargs*)

Add a suggestions request to the search.

Parameters

- **name** – name of the suggestion
- **text** – text to suggest on

All keyword arguments will be added to the suggestions body. For example:

```
s = Search()
s = s.suggest('suggestion-1', 'Elasticsearch', term={'field': 'body'})
```

to_dict (*count=False*, ***kwargs*)

Serialize the search into the dictionary that will be sent over as the request's body.

Parameters **count** – a flag to specify if we are interested in a body for count - no aggregations, no pagination bounds etc.

All additional keyword arguments will be included into the dictionary.

update_from_dict (*d*)

Apply options from a serialized body to the current instance. Modifies the object in-place. Used mostly by `from_dict`.

5.2.9 pandagg.utils module

class `pandagg.utils.DslMeta` (*name*, *bases*, *attrs*)

Bases: `type`

Base Metaclass for `DslBase` subclasses that builds a registry of all classes for given `DslBase` subclass (== all the query types for the `Query` subclass of `DslBase`).

It then uses the information from that registry (as well as *name* and *deserializer* attributes from the base class) to construct any subclass based on it's name.

classmethod `get_dsl_type` (*name*)

`pandagg.utils.equal_queries` (*d1*, *d2*)

Compares if two queries are equivalent (do not consider nested list orders).

`pandagg.utils.equal_search` (*s1*, *s2*)

`pandagg.utils.get_dsl_class` (*cls*, *name*)

`pandagg.utils.ordered` (*obj*)

5.3 Module contents

We want to make contributing to this project as easy and transparent as possible.

6.1 Our Development Process

We use github to host code, to track issues and feature requests, as well as accept pull requests.

6.2 Pull Requests

We actively welcome your pull requests.

1. Fork the repo and create your branch from `master`.
2. If you've added code that should be tested, add tests.
3. If you've changed APIs, update the documentation.
4. Ensure the test suite passes.
5. Make sure your code lints.

6.3 Any contributions you make will be under the MIT Software License

In short, when you submit code changes, your submissions are understood to be under the same [MIT License](#) that covers the project. Feel free to contact the maintainers if that's a concern.

6.4 Issues

We use GitHub issues to track public bugs. Please ensure your description is clear and has sufficient instructions to be able to reproduce the issue.

6.5 Report bugs using Github's issues

We use GitHub issues to track public bugs. Report a bug by [opening a new issue](#); it's that easy!

6.6 Write bug reports with detail, background, and sample code

Great Bug Reports tend to have:

- A quick summary and/or background
- Steps to reproduce
 - Be specific!
 - Give sample code if you can.
- What you expected would happen
- What actually happens
- Notes (possibly including why you think this might be happening, or stuff you tried that didn't work)

6.7 License

By contributing, you agree that your contributions will be licensed under its MIT License.

6.8 References

This document was adapted from the open-source contribution guidelines of [briandk's gist](#)

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 7

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 8

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 9

License

pandagg is made available under the MIT License. For more details, see [LICENSE.txt](#).

CHAPTER 10

Contributing

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

p

- `pandagg`, 78
- `pandagg.aggs`, 52
- `pandagg.connections`, 58
- `pandagg.discovery`, 59
- `pandagg.exceptions`, 59
- `pandagg.interactive`, 18
- `pandagg.interactive.mapping`, 17
- `pandagg.interactive.response`, 17
- `pandagg.mapping`, 59
- `pandagg.node`, 37
- `pandagg.node.aggs`, 26
- `pandagg.node.aggs.abstract`, 18
- `pandagg.node.aggs.bucket`, 20
- `pandagg.node.aggs.metric`, 23
- `pandagg.node.aggs.pipeline`, 24
- `pandagg.node.mapping`, 31
- `pandagg.node.mapping.abstract`, 26
- `pandagg.node.mapping.field_datatypes`, 27
- `pandagg.node.mapping.meta_fields`, 30
- `pandagg.node.query`, 37
- `pandagg.node.query.abstract`, 31
- `pandagg.node.query.compound`, 32
- `pandagg.node.query.full_text`, 33
- `pandagg.node.query.geo`, 34
- `pandagg.node.query.joining`, 34
- `pandagg.node.query.shape`, 35
- `pandagg.node.query.span`, 35
- `pandagg.node.query.specialized`, 35
- `pandagg.node.query.specialized_compound`, 35
- `pandagg.node.query.term_level`, 35
- `pandagg.node.response`, 37
- `pandagg.node.response.bucket`, 37
- `pandagg.node.types`, 37
- `pandagg.query`, 64
- `pandagg.response`, 69
- `pandagg.search`, 71
- `pandagg.tree`, 52
- `pandagg.tree.aggs`, 44
- `pandagg.tree.aggs.aggs`, 37
- `pandagg.tree.aggs.bucket`, 41
- `pandagg.tree.aggs.metric`, 42
- `pandagg.tree.aggs.pipeline`, 43
- `pandagg.tree.mapping`, 50
- `pandagg.tree.query`, 50
- `pandagg.tree.query.abstract`, 44
- `pandagg.tree.query.compound`, 47
- `pandagg.tree.query.full_text`, 47
- `pandagg.tree.query.geo`, 48
- `pandagg.tree.query.joining`, 48
- `pandagg.tree.query.shape`, 48
- `pandagg.tree.query.span`, 49
- `pandagg.tree.query.specialized`, 49
- `pandagg.tree.query.specialized_compound`, 49
- `pandagg.tree.query.term_level`, 49
- `pandagg.tree.response`, 51
- `pandagg.utils`, 77

A

AbsentMappingFieldError, 59
 AbstractLeafAgg (class in *pandagg.tree.aggs.aggs*), 37
 AbstractParentAgg (class in *pandagg.tree.aggs.aggs*), 37
 AbstractSingleFieldQueryClause (class in *pandagg.node.query.abstract*), 31
 add() (*pandagg.search.MultiSearch* method), 71
 add_connection() (*pandagg.connections.Connections* method), 58
 AggNode (class in *pandagg.node.aggs.abstract*), 18
 Aggregations (class in *pandagg.response*), 69
 Aggs (class in *pandagg.aggs*), 52
 Aggs (class in *pandagg.tree.aggs.aggs*), 38
 aggs() (*pandagg.aggs.Aggs* method), 53
 aggs() (*pandagg.search.Search* method), 71
 aggs() (*pandagg.tree.aggs.aggs.Aggs* method), 38
 AggsResponseTree (class in *pandagg.tree.response*), 51
 Alias (class in *pandagg.mapping*), 63
 Alias (class in *pandagg.node.mapping.field_datatypes*), 27
 applied_nested_path_at_node() (*pandagg.aggs.Aggs* method), 53
 applied_nested_path_at_node() (*pandagg.query.Query* method), 65
 applied_nested_path_at_node() (*pandagg.tree.aggs.aggs.Aggs* method), 39
 applied_nested_path_at_node() (*pandagg.tree.query.abstract.Query* method), 45
 attr_name (*pandagg.node.response.bucket.Bucket* attribute), 37
 Avg (class in *pandagg.aggs*), 56
 Avg (class in *pandagg.node.aggs.metric*), 23
 Avg (class in *pandagg.tree.aggs.metric*), 42
 AvgBucket (class in *pandagg.aggs*), 57
 AvgBucket (class in *pandagg.node.aggs.pipeline*), 24

AvgBucket (class in *pandagg.tree.aggs.pipeline*), 43

B

Binary (class in *pandagg.mapping*), 61
 Binary (class in *pandagg.node.mapping.field_datatypes*), 27
 BLACKLISTED_MAPPING_TYPES (*pandagg.node.aggs.abstract.AggNode* attribute), 18
 BLACKLISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.Missing* attribute), 22
 BLACKLISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.Terms* attribute), 22
 BLACKLISTED_MAPPING_TYPES (*pandagg.node.aggs.metric.ValueCount* attribute), 24
 body (*pandagg.node.mapping.abstract.Field* attribute), 26
 Bool (class in *pandagg.node.query.compound*), 32
 Bool (class in *pandagg.query*), 68
 Bool (class in *pandagg.tree.query.compound*), 47
 bool() (*pandagg.query.Query* method), 65
 bool() (*pandagg.search.Search* method), 72
 bool() (*pandagg.tree.query.abstract.Query* method), 45
 Boolean (class in *pandagg.mapping*), 61
 Boolean (class in *pandagg.node.mapping.field_datatypes*), 27
 boost() (*pandagg.query.Query* method), 65
 boost() (*pandagg.tree.query.abstract.Query* method), 45
 Boosting (class in *pandagg.node.query.compound*), 32
 Boosting (class in *pandagg.query*), 68
 Boosting (class in *pandagg.tree.query.compound*), 47
 Bucket (class in *pandagg.node.response.bucket*), 37
 bucket_properties() (*pandagg.tree.response.AggsResponseTree* method), 51

- BucketAggNode (class *pandagg.node.aggs.abstract*), 19
- BucketScript (class in *pandagg.aggs*), 58
- BucketScript (class in *pandagg.node.aggs.pipeline*), 24
- BucketScript (class in *pandagg.tree.aggs.pipeline*), 43
- BucketSelector (class in *pandagg.aggs*), 58
- BucketSelector (class *pandagg.node.aggs.pipeline*), 25
- BucketSelector (class *pandagg.tree.aggs.pipeline*), 43
- BucketSort (class in *pandagg.aggs*), 58
- BucketSort (class in *pandagg.node.aggs.pipeline*), 25
- BucketSort (class in *pandagg.tree.aggs.pipeline*), 43
- Byte (class in *pandagg.mapping*), 60
- Byte (class in *pandagg.node.mapping.field_datatypes*), 27
- C**
- Cardinality (class in *pandagg.aggs*), 56
- Cardinality (class in *pandagg.node.aggs.metric*), 23
- Cardinality (class in *pandagg.tree.aggs.metric*), 42
- Common (class in *pandagg.node.query.full_text*), 33
- Common (class in *pandagg.query*), 68
- Common (class in *pandagg.tree.query.full_text*), 47
- Completion (class in *pandagg.mapping*), 62
- Completion (class *pandagg.node.mapping.field_datatypes*), 27
- Composite (class in *pandagg.node.aggs.bucket*), 20
- Composite (class in *pandagg.tree.aggs.bucket*), 41
- Compound (class in *pandagg.tree.query.abstract*), 44
- CompoundClause (class *pandagg.node.query.compound*), 32
- configure() (*pandagg.connections.Connections* method), 58
- Connections (class in *pandagg.connections*), 58
- constant_score() (*pandagg.query.Query* method), 65
- constant_score() (*pandagg.tree.query.abstract.Query* method), 45
- ConstantScore (class *pandagg.node.query.compound*), 33
- ConstantScore (class in *pandagg.query*), 68
- ConstantScore (class *pandagg.tree.query.compound*), 47
- count() (*pandagg.search.Search* method), 72
- create_connection() (*pandagg.connections.Connections* method), 58
- CumulativeSum (class in *pandagg.aggs*), 58
- CumulativeSum (class *pandagg.node.aggs.pipeline*), 25
- CumulativeSum (class in *pandagg.tree.aggs.pipeline*), 43
- D**
- Date (class in *pandagg.mapping*), 61
- Date (class in *pandagg.node.mapping.field_datatypes*), 27
- DateHistogram (class in *pandagg.aggs*), 55
- DateHistogram (class in *pandagg.node.aggs.bucket*), 20
- DateHistogram (class in *pandagg.tree.aggs.bucket*), 41
- DateNanos (class in *pandagg.mapping*), 61
- DateNanos (class *pandagg.node.mapping.field_datatypes*), 27
- DateRange (class in *pandagg.mapping*), 61
- DateRange (class in *pandagg.node.aggs.bucket*), 21
- DateRange (class *pandagg.node.mapping.field_datatypes*), 27
- DateRange (class in *pandagg.tree.aggs.bucket*), 41
- deepest_linear_bucket_agg (*pandagg.aggs.Aggs* attribute), 53
- deepest_linear_bucket_agg (*pandagg.tree.aggs.aggs.Aggs* attribute), 39
- DEFAULT_OTHER_KEY (*pandagg.node.aggs.bucket.Filters* attribute), 21
- delete() (*pandagg.search.Search* method), 72
- DenseVector (class in *pandagg.mapping*), 63
- DenseVector (class *pandagg.node.mapping.field_datatypes*), 27
- Derivative (class in *pandagg.aggs*), 57
- Derivative (class in *pandagg.node.aggs.pipeline*), 25
- Derivative (class in *pandagg.tree.aggs.pipeline*), 43
- dis_max() (*pandagg.query.Query* method), 65
- dis_max() (*pandagg.tree.query.abstract.Query* method), 45
- discover() (in module *pandagg.discovery*), 59
- DisMax (class in *pandagg.node.query.compound*), 33
- DisMax (class in *pandagg.query*), 68
- DisMax (class in *pandagg.tree.query.compound*), 47
- DistanceFeature (class *pandagg.node.query.specialized*), 35
- DistanceFeature (class in *pandagg.query*), 69
- DistanceFeature (class *pandagg.tree.query.specialized*), 49
- Double (class in *pandagg.mapping*), 60
- Double (class in *pandagg.node.mapping.field_datatypes*), 27
- DoubleRange (class in *pandagg.mapping*), 61

DoubleRange (class
pandagg.node.mapping.field_datatypes),
28

DslMeta (class in pandagg.utils), 77

E

equal_queries() (in module pandagg.utils), 77

equal_search() (in module pandagg.utils), 77

exclude() (pandagg.search.Search method), 72

execute() (pandagg.search.MultiSearch method), 71

execute() (pandagg.search.Search method), 72

Exists (class in pandagg.node.query.term_level), 35

Exists (class in pandagg.query), 66

Exists (class in pandagg.tree.query.term_level), 49

ExtendedStats (class in pandagg.aggs), 56

ExtendedStats (class in pandagg.node.aggs.metric),
23

ExtendedStats (class in pandagg.tree.aggs.metric),
42

ExtendedStatsBucket (class in pandagg.aggs), 57

ExtendedStatsBucket (class in
pandagg.node.aggs.pipeline), 25

ExtendedStatsBucket (class in
pandagg.tree.aggs.pipeline), 43

extract_bucket_value()
(pandagg.node.aggs.abstract.AggNode class
method), 18

extract_bucket_value()
(pandagg.node.aggs.abstract.ShadowRoot
class method), 20

extract_buckets()
(pandagg.node.aggs.abstract.AggNode
method), 18

extract_buckets()
(pandagg.node.aggs.abstract.BucketAggNode
method), 19

extract_buckets()
(pandagg.node.aggs.abstract.MetricAgg
method), 19

extract_buckets()
(pandagg.node.aggs.abstract.MultipleBucketAgg
method), 19

extract_buckets()
(pandagg.node.aggs.abstract.UniqueBucketAgg
method), 20

F

Field (class in pandagg.node.mapping.abstract), 26

FieldNames (class in pandagg.mapping), 64

FieldNames (class
pandagg.node.mapping.meta_fields), 30

FieldOrScriptMetricAgg (class
pandagg.node.aggs.abstract), 19

Filter (class in pandagg.aggs), 56

in Filter (class in pandagg.node.aggs.bucket), 21

Filter (class in pandagg.tree.aggs.bucket), 41

filter() (pandagg.query.Query method), 65

filter() (pandagg.search.Search method), 72

filter() (pandagg.tree.query.abstract.Query
method), 45

Filters (class in pandagg.aggs), 55

Filters (class in pandagg.node.aggs.bucket), 21

Filters (class in pandagg.tree.aggs.bucket), 41

FlatFieldQueryClause (class in
pandagg.node.query.abstract), 31

Flattened (class in pandagg.mapping), 63

Flattened (class in
pandagg.node.mapping.field_datatypes),
28

Float (class in pandagg.mapping), 61

Float (class in pandagg.node.mapping.field_datatypes),
28

FloatRange (class in pandagg.mapping), 61

FloatRange (class in
pandagg.node.mapping.field_datatypes),
28

from_dict() (pandagg.search.Search class method),
72

from_key (pandagg.node.aggs.bucket.Range attribute),
22

function_score() (pandagg.query.Query method),
65

function_score() (pandagg.tree.query.abstract.Query
method), 45

FunctionScore (class in
pandagg.node.query.compound), 33

FunctionScore (class in pandagg.query), 68

FunctionScore (class in
pandagg.tree.query.compound), 47

Fuzzy (class in pandagg.node.query.term_level), 36

Fuzzy (class in pandagg.query), 66

Fuzzy (class in pandagg.tree.query.term_level), 49

G

GeoBound (class in pandagg.aggs), 57

GeoBound (class in pandagg.node.aggs.metric), 23

GeoBound (class in pandagg.tree.aggs.metric), 42

GeoBoundingBox (class in pandagg.node.query.geo),
34

GeoBoundingBox (class in pandagg.query), 69

GeoBoundingBox (class in pandagg.tree.query.geo),
48

GeoCentroid (class in pandagg.aggs), 57

in GeoCentroid (class in pandagg.node.aggs.metric), 23

GeoCentroid (class in pandagg.tree.aggs.metric), 42

in GeoDistance (class in pandagg.node.query.geo), 34

GeoDistance (class in pandagg.query), 69

GeoDistance (class in pandagg.tree.query.geo), 48

- GeoPoint (class in pandagg.mapping), 61
 - GeoPoint (class in pandagg.node.mapping.field_datatypes), 28
 - GeoPolygone (class in pandagg.node.query.geo), 34
 - GeoPolygone (class in pandagg.query), 69
 - GeoPolygone (class in pandagg.tree.query.geo), 48
 - GeoShape (class in pandagg.mapping), 62
 - GeoShape (class in pandagg.node.mapping.field_datatypes), 28
 - GeoShape (class in pandagg.node.query.geo), 34
 - GeoShape (class in pandagg.query), 68
 - GeoShape (class in pandagg.tree.query.geo), 48
 - get () (pandagg.mapping.Mapping method), 59
 - get () (pandagg.response.Aggregations method), 69
 - get () (pandagg.tree.mapping.Mapping method), 50
 - get_bucket_filter () (pandagg.interactive.response.IResponse method), 17
 - get_bucket_filter () (pandagg.tree.response.AggsResponseTree method), 51
 - get_connection () (pandagg.connections.Connections method), 58
 - get_dsl_class () (in module pandagg.utils), 77
 - get_dsl_class () (pandagg.node.mapping.abstract.UnnamedField class method), 27
 - get_dsl_type () (pandagg.utils.DslMeta class method), 77
 - get_filter () (pandagg.node.aggs.abstract.AgNode method), 18
 - get_filter () (pandagg.node.aggs.abstract.BucketAgNode method), 19
 - get_filter () (pandagg.node.aggs.abstract.MetricAg method), 19
 - get_filter () (pandagg.node.aggs.abstract.MultipleBucketAg method), 19
 - get_filter () (pandagg.node.aggs.abstract.Pipeline method), 20
 - get_filter () (pandagg.node.aggs.abstract.ShadowRoot method), 20
 - get_filter () (pandagg.node.aggs.abstract.UniqueBucketAg method), 20
 - get_filter () (pandagg.node.aggs.bucket.Composite method), 20
 - get_filter () (pandagg.node.aggs.bucket.DateHistogram method), 21
 - get_filter () (pandagg.node.aggs.bucket.Filter method), 21
 - get_filter () (pandagg.node.aggs.bucket.Filters method), 21
 - get_filter () (pandagg.node.aggs.bucket.Global method), 21
 - get_filter () (pandagg.node.aggs.bucket.Histogram method), 21
 - get_filter () (pandagg.node.aggs.bucket.Missing method), 22
 - get_filter () (pandagg.node.aggs.bucket.Nested method), 22
 - get_filter () (pandagg.node.aggs.bucket.Range method), 22
 - get_filter () (pandagg.node.aggs.bucket.ReverseNested method), 22
 - get_filter () (pandagg.node.aggs.bucket.Terms method), 22
 - Global (class in pandagg.aggs), 56
 - Global (class in pandagg.node.aggs.bucket), 21
 - Global (class in pandagg.tree.aggs.bucket), 41
 - groupby () (pandagg.aggs.Aggs method), 53
 - groupby () (pandagg.search.Search method), 73
 - groupby () (pandagg.tree.aggs.aggs.Aggs method), 39
- ## H
- HalfFloat (class in pandagg.mapping), 60
 - HalfFloat (class in pandagg.node.mapping.field_datatypes), 28
 - has_child () (pandagg.query.Query method), 65
 - has_child () (pandagg.tree.query.abstract.Query method), 45
 - has_parent () (pandagg.query.Query method), 65
 - has_parent () (pandagg.tree.query.abstract.Query method), 45
 - HasChild (class in pandagg.node.query.joining), 34
 - HasChild (class in pandagg.query), 68
 - HasChild (class in pandagg.tree.query.joining), 48
 - HasParent (class in pandagg.node.query.joining), 34
 - HasParent (class in pandagg.query), 68
 - HasParent (class in pandagg.tree.query.joining), 48
 - highlight () (pandagg.search.Search method), 74
 - highlight_options () (pandagg.search.Search method), 74
 - Histogram (class in pandagg.aggs), 55
 - Histogram (class in pandagg.mapping), 63
 - Histogram (class in pandagg.node.aggs.bucket), 21
 - Histogram (class in pandagg.tree.aggs.bucket), 41
 - Hit (class in pandagg.response), 70
 - Hits (class in pandagg.response), 70
- ## I
- Id (class in pandagg.mapping), 63
 - Id (class in pandagg.node.mapping.meta_fields), 30
 - Ids (class in pandagg.node.query.term_level), 36
 - Ids (class in pandagg.query), 66
 - Ids (class in pandagg.tree.query.term_level), 49
 - Ignored (class in pandagg.mapping), 64

- Ignored (class in *pandagg.node.mapping.meta_fields*), 30
- IMapping (class in *pandagg.interactive.mapping*), 17
- IMapping (class in *pandagg.mapping*), 60
- IMPLICIT_KEYED (pandagg.node.aggs.abstract.MultipleBucketAgg attribute), 19
- IMPLICIT_KEYED (pandagg.node.aggs.bucket.Filters attribute), 21
- Index (class in *pandagg.discovery*), 59
- Index (class in *pandagg.mapping*), 63
- Index (class in *pandagg.node.mapping.meta_fields*), 30
- index() (pandagg.search.Request method), 71
- Indices (class in *pandagg.discovery*), 59
- Integer (class in *pandagg.mapping*), 60
- Integer (class in *pandagg.node.mapping.field_datatypes*), 28
- IntegerRange (class in *pandagg.mapping*), 61
- IntegerRange (class in *pandagg.node.mapping.field_datatypes*), 28
- Intervals (class in *pandagg.node.query.full_text*), 33
- Intervals (class in *pandagg.query*), 67
- Intervals (class in *pandagg.tree.query.full_text*), 47
- InvalidAggregation, 59
- InvalidOperationMappingFieldError, 59
- IP (class in *pandagg.mapping*), 62
- IP (class in *pandagg.node.mapping.field_datatypes*), 28
- IResponse (class in *pandagg.interactive.response*), 17
- ## J
- Join (class in *pandagg.mapping*), 62
- Join (class in *pandagg.node.mapping.field_datatypes*), 29
- ## K
- KEY (pandagg.aggs.Avg attribute), 56
- KEY (pandagg.aggs.AvgBucket attribute), 57
- KEY (pandagg.aggs.BucketScript attribute), 58
- KEY (pandagg.aggs.BucketSelector attribute), 58
- KEY (pandagg.aggs.BucketSort attribute), 58
- KEY (pandagg.aggs.Cardinality attribute), 56
- KEY (pandagg.aggs.CumulativeSum attribute), 58
- KEY (pandagg.aggs.DateHistogram attribute), 55
- KEY (pandagg.aggs.Derivative attribute), 57
- KEY (pandagg.aggs.ExtendedStats attribute), 56
- KEY (pandagg.aggs.ExtendedStatsBucket attribute), 57
- KEY (pandagg.aggs.Filter attribute), 56
- KEY (pandagg.aggs.Filters attribute), 55
- KEY (pandagg.aggs.GeoBound attribute), 57
- KEY (pandagg.aggs.GeoCentroid attribute), 57
- KEY (pandagg.aggs.Global attribute), 56
- KEY (pandagg.aggs.Histogram attribute), 55
- KEY (pandagg.aggs.Max attribute), 56
- KEY (pandagg.aggs.MaxBucket attribute), 57
- KEY (pandagg.aggs.Min attribute), 56
- KEY (pandagg.aggs.MinBucket attribute), 57
- KEY (pandagg.aggs.Missing attribute), 56
- KEY (pandagg.aggs.MovingAvg attribute), 58
- KEY (pandagg.aggs.Nested attribute), 56
- KEY (pandagg.aggs.PercentileRanks attribute), 57
- KEY (pandagg.aggs.Percentiles attribute), 56
- KEY (pandagg.aggs.PercentilesBucket attribute), 57
- KEY (pandagg.aggs.Range attribute), 55
- KEY (pandagg.aggs.ReverseNested attribute), 56
- KEY (pandagg.aggs.SerialDiff attribute), 58
- KEY (pandagg.aggs.Stats attribute), 56
- KEY (pandagg.aggs.StatsBucket attribute), 57
- KEY (pandagg.aggs.Sum attribute), 56
- KEY (pandagg.aggs.SumBucket attribute), 57
- KEY (pandagg.aggs.Terms attribute), 55
- KEY (pandagg.aggs.TopHits attribute), 57
- KEY (pandagg.aggs.ValueCount attribute), 57
- KEY (pandagg.mapping.Alias attribute), 63
- KEY (pandagg.mapping.Binary attribute), 61
- KEY (pandagg.mapping.Boolean attribute), 61
- KEY (pandagg.mapping.Byte attribute), 60
- KEY (pandagg.mapping.Completion attribute), 62
- KEY (pandagg.mapping.Date attribute), 61
- KEY (pandagg.mapping.DateNanos attribute), 61
- KEY (pandagg.mapping.DateRange attribute), 61
- KEY (pandagg.mapping.DenseVector attribute), 63
- KEY (pandagg.mapping.Double attribute), 60
- KEY (pandagg.mapping.DoubleRange attribute), 61
- KEY (pandagg.mapping.FieldNames attribute), 64
- KEY (pandagg.mapping.Flattened attribute), 63
- KEY (pandagg.mapping.Float attribute), 61
- KEY (pandagg.mapping.FloatRange attribute), 61
- KEY (pandagg.mapping.GeoPoint attribute), 61
- KEY (pandagg.mapping.GeoShape attribute), 62
- KEY (pandagg.mapping.HalfFloat attribute), 60
- KEY (pandagg.mapping.Histogram attribute), 63
- KEY (pandagg.mapping.Id attribute), 64
- KEY (pandagg.mapping.Ignored attribute), 64
- KEY (pandagg.mapping.Index attribute), 63
- KEY (pandagg.mapping.Integer attribute), 60
- KEY (pandagg.mapping.IntegerRange attribute), 61
- KEY (pandagg.mapping.IP attribute), 62
- KEY (pandagg.mapping.Join attribute), 62
- KEY (pandagg.mapping.Keyword attribute), 60
- KEY (pandagg.mapping.Long attribute), 60
- KEY (pandagg.mapping.LongRange attribute), 61
- KEY (pandagg.mapping.MapperAnnotatedText attribute), 62
- KEY (pandagg.mapping.MapperMurMur3 attribute), 62
- KEY (pandagg.mapping.Mapping attribute), 59
- KEY (pandagg.mapping.Meta attribute), 64
- KEY (pandagg.mapping.Nested attribute), 61
- KEY (pandagg.mapping.Object attribute), 61

- KEY (*pandagg.mapping.Perculator attribute*), 62
- KEY (*pandagg.mapping.RankFeature attribute*), 62
- KEY (*pandagg.mapping.RankFeatures attribute*), 63
- KEY (*pandagg.mapping.Routing attribute*), 64
- KEY (*pandagg.mapping.ScaledFloat attribute*), 60
- KEY (*pandagg.mapping.SearchAsYouType attribute*), 63
- KEY (*pandagg.mapping.Shape attribute*), 63
- KEY (*pandagg.mapping.Short attribute*), 60
- KEY (*pandagg.mapping.Size attribute*), 64
- KEY (*pandagg.mapping.Source attribute*), 64
- KEY (*pandagg.mapping.SparseVector attribute*), 63
- KEY (*pandagg.mapping.Text attribute*), 60
- KEY (*pandagg.mapping.TokenCount attribute*), 62
- KEY (*pandagg.mapping.Type attribute*), 63
- KEY (*pandagg.node.aggs.abstract.AggNode attribute*), 18
- KEY (*pandagg.node.aggs.abstract.ScriptPipeline attribute*), 20
- KEY (*pandagg.node.aggs.abstract.ShadowRoot attribute*), 20
- KEY (*pandagg.node.aggs.bucket.Composite attribute*), 20
- KEY (*pandagg.node.aggs.bucket.DateHistogram attribute*), 20
- KEY (*pandagg.node.aggs.bucket.DateRange attribute*), 21
- KEY (*pandagg.node.aggs.bucket.Filter attribute*), 21
- KEY (*pandagg.node.aggs.bucket.Filters attribute*), 21
- KEY (*pandagg.node.aggs.bucket.Global attribute*), 21
- KEY (*pandagg.node.aggs.bucket.Histogram attribute*), 21
- KEY (*pandagg.node.aggs.bucket.Missing attribute*), 22
- KEY (*pandagg.node.aggs.bucket.Nested attribute*), 22
- KEY (*pandagg.node.aggs.bucket.Range attribute*), 22
- KEY (*pandagg.node.aggs.bucket.ReverseNested attribute*), 22
- KEY (*pandagg.node.aggs.bucket.Terms attribute*), 22
- KEY (*pandagg.node.aggs.metric.Avg attribute*), 23
- KEY (*pandagg.node.aggs.metric.Cardinality attribute*), 23
- KEY (*pandagg.node.aggs.metric.ExtendedStats attribute*), 23
- KEY (*pandagg.node.aggs.metric.GeoBound attribute*), 23
- KEY (*pandagg.node.aggs.metric.GeoCentroid attribute*), 23
- KEY (*pandagg.node.aggs.metric.Max attribute*), 23
- KEY (*pandagg.node.aggs.metric.Min attribute*), 23
- KEY (*pandagg.node.aggs.metric.PercentileRanks attribute*), 23
- KEY (*pandagg.node.aggs.metric.Percentiles attribute*), 24
- KEY (*pandagg.node.aggs.metric.Stats attribute*), 24
- KEY (*pandagg.node.aggs.metric.Sum attribute*), 24
- KEY (*pandagg.node.aggs.metric.TopHits attribute*), 24
- KEY (*pandagg.node.aggs.metric.ValueCount attribute*), 24
- KEY (*pandagg.node.aggs.pipeline.AvgBucket attribute*), 24
- KEY (*pandagg.node.aggs.pipeline.BucketScript attribute*), 24
- KEY (*pandagg.node.aggs.pipeline.BucketSelector attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.BucketSort attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.CumulativeSum attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.Derivative attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.ExtendedStatsBucket attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.MaxBucket attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.MinBucket attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.MovingAvg attribute*), 25
- KEY (*pandagg.node.aggs.pipeline.PercentilesBucket attribute*), 26
- KEY (*pandagg.node.aggs.pipeline.SerialDiff attribute*), 26
- KEY (*pandagg.node.aggs.pipeline.StatsBucket attribute*), 26
- KEY (*pandagg.node.aggs.pipeline.SumBucket attribute*), 26
- KEY (*pandagg.node.mapping.abstract.ShadowRoot attribute*), 26
- KEY (*pandagg.node.mapping.abstract.UnnamedComplexField attribute*), 26
- KEY (*pandagg.node.mapping.abstract.UnnamedField attribute*), 26
- KEY (*pandagg.node.mapping.abstract.UnnamedRegularField attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.Alias attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.Binary attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.Boolean attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.Byte attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.Completion attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.Date attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.DateNanos attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.DateRange attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.DenseVector attribute*), 27
- KEY (*pandagg.node.mapping.field_datatypes.Double attribute*), 28

KEY (<i>pandagg.node.mapping.field_datatypes.DoubleRange attribute</i>), 28	KEY (<i>pandagg.node.mapping.field_datatypes.Text attribute</i>), 30
KEY (<i>pandagg.node.mapping.field_datatypes.Flattened attribute</i>), 28	KEY (<i>pandagg.node.mapping.field_datatypes.TokenCount attribute</i>), 30
KEY (<i>pandagg.node.mapping.field_datatypes.Float attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.FieldNames attribute</i>), 30
KEY (<i>pandagg.node.mapping.field_datatypes.FloatRange attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Id attribute</i>), 30
KEY (<i>pandagg.node.mapping.field_datatypes.GeoPoint attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Ignored attribute</i>), 30
KEY (<i>pandagg.node.mapping.field_datatypes.GeoShape attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Index attribute</i>), 31
KEY (<i>pandagg.node.mapping.field_datatypes.HalfFloat attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Meta attribute</i>), 31
KEY (<i>pandagg.node.mapping.field_datatypes.Histogram attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Routing attribute</i>), 31
KEY (<i>pandagg.node.mapping.field_datatypes.Integer attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Size attribute</i>), 31
KEY (<i>pandagg.node.mapping.field_datatypes.IntegerRange attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Source attribute</i>), 31
KEY (<i>pandagg.node.mapping.field_datatypes.IP attribute</i>), 28	KEY (<i>pandagg.node.mapping.meta_fields.Type attribute</i>), 31
KEY (<i>pandagg.node.mapping.field_datatypes.Join attribute</i>), 29	KEY (<i>pandagg.node.query.abstract.QueryClause attribute</i>), 32
KEY (<i>pandagg.node.mapping.field_datatypes.Keyword attribute</i>), 29	KEY (<i>pandagg.node.query.compound.Bool attribute</i>), 32
KEY (<i>pandagg.node.mapping.field_datatypes.Long attribute</i>), 29	KEY (<i>pandagg.node.query.compound.Boosting attribute</i>), 32
KEY (<i>pandagg.node.mapping.field_datatypes.LongRange attribute</i>), 29	KEY (<i>pandagg.node.query.compound.ConstantScore attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.MapperAnnotatedText attribute</i>), 29	KEY (<i>pandagg.node.query.compound.DisMax attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.MapperMurMur3 attribute</i>), 29	KEY (<i>pandagg.node.query.compound.FunctionScore attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.Nested attribute</i>), 29	KEY (<i>pandagg.node.query.full_text.Common attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.Object attribute</i>), 29	KEY (<i>pandagg.node.query.full_text.Intervals attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.Perculator attribute</i>), 29	KEY (<i>pandagg.node.query.full_text.Match attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.RankFeature attribute</i>), 29	KEY (<i>pandagg.node.query.full_text.MatchBoolPrefix attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.RankFeatures attribute</i>), 29	KEY (<i>pandagg.node.query.full_text.MatchPhrase attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.ScaledFloat attribute</i>), 30	KEY (<i>pandagg.node.query.full_text.MatchPhrasePrefix attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.SearchAsYouType attribute</i>), 30	KEY (<i>pandagg.node.query.full_text.MultiMatch attribute</i>), 33
KEY (<i>pandagg.node.mapping.field_datatypes.Shape attribute</i>), 30	KEY (<i>pandagg.node.query.full_text.QueryString attribute</i>), 34
KEY (<i>pandagg.node.mapping.field_datatypes.Short attribute</i>), 30	KEY (<i>pandagg.node.query.full_text.SimpleQueryString attribute</i>), 34
KEY (<i>pandagg.node.mapping.field_datatypes.SparseVector attribute</i>), 30	KEY (<i>pandagg.node.query.geo.GeoBoundingBox attribute</i>), 34
	KEY (<i>pandagg.node.query.geo.GeoDistance attribute</i>), 34

- KEY (*pandagg.node.query.geo.GeoPolygone* attribute), 34
- KEY (*pandagg.node.query.geo.GeoShape* attribute), 34
- KEY (*pandagg.node.query.joining.HasChild* attribute), 34
- KEY (*pandagg.node.query.joining.HasParent* attribute), 34
- KEY (*pandagg.node.query.joining.Nested* attribute), 34
- KEY (*pandagg.node.query.joining.ParentId* attribute), 34
- KEY (*pandagg.node.query.shape.Shape* attribute), 35
- KEY (*pandagg.node.query.specialized.DistanceFeature* attribute), 35
- KEY (*pandagg.node.query.specialized.MoreLikeThis* attribute), 35
- KEY (*pandagg.node.query.specialized.Percolate* attribute), 35
- KEY (*pandagg.node.query.specialized.RankFeature* attribute), 35
- KEY (*pandagg.node.query.specialized.Script* attribute), 35
- KEY (*pandagg.node.query.specialized.Wrapper* attribute), 35
- KEY (*pandagg.node.query.specialized_compound.PinnedQuery* attribute), 35
- KEY (*pandagg.node.query.specialized_compound.ScriptScore* attribute), 35
- KEY (*pandagg.node.query.term_level.Exists* attribute), 35
- KEY (*pandagg.node.query.term_level.Fuzzy* attribute), 36
- KEY (*pandagg.node.query.term_level.Ids* attribute), 36
- KEY (*pandagg.node.query.term_level.Prefix* attribute), 36
- KEY (*pandagg.node.query.term_level.Range* attribute), 36
- KEY (*pandagg.node.query.term_level.Regexp* attribute), 36
- KEY (*pandagg.node.query.term_level.Term* attribute), 36
- KEY (*pandagg.node.query.term_level.Terms* attribute), 36
- KEY (*pandagg.node.query.term_level.TermsSet* attribute), 36
- KEY (*pandagg.node.query.term_level.Type* attribute), 36
- KEY (*pandagg.node.query.term_level.Wildcard* attribute), 36
- KEY (*pandagg.query.Bool* attribute), 68
- KEY (*pandagg.query.Boosting* attribute), 68
- KEY (*pandagg.query.Common* attribute), 68
- KEY (*pandagg.query.ConstantScore* attribute), 68
- KEY (*pandagg.query.DisMax* attribute), 68
- KEY (*pandagg.query.DistanceFeature* attribute), 69
- KEY (*pandagg.query.Exists* attribute), 66
- KEY (*pandagg.query.FunctionScore* attribute), 68
- KEY (*pandagg.query.Fuzzy* attribute), 66
- KEY (*pandagg.query.GeoBoundingBox* attribute), 69
- KEY (*pandagg.query.GeoDistance* attribute), 69
- KEY (*pandagg.query.GeoPolygone* attribute), 69
- KEY (*pandagg.query.GeoShape* attribute), 69
- KEY (*pandagg.query.HasChild* attribute), 68
- KEY (*pandagg.query.HasParent* attribute), 68
- KEY (*pandagg.query.Ids* attribute), 66
- KEY (*pandagg.query.Intervals* attribute), 67
- KEY (*pandagg.query.Match* attribute), 67
- KEY (*pandagg.query.MatchBoolPrefix* attribute), 67
- KEY (*pandagg.query.MatchPhrase* attribute), 67
- KEY (*pandagg.query.MatchPhrasePrefix* attribute), 67
- KEY (*pandagg.query.MoreLikeThis* attribute), 69
- KEY (*pandagg.query.MultiMatch* attribute), 67
- KEY (*pandagg.query.Nested* attribute), 68
- KEY (*pandagg.query.ParentId* attribute), 68
- KEY (*pandagg.query.Percolate* attribute), 69
- KEY (*pandagg.query.PinnedQuery* attribute), 69
- KEY (*pandagg.query.Prefix* attribute), 67
- KEY (*pandagg.query.Query* attribute), 65
- KEY (*pandagg.query.QueryString* attribute), 68
- KEY (*pandagg.query.Range* attribute), 67
- KEY (*pandagg.query.RankFeature* attribute), 69
- KEY (*pandagg.query.Regexp* attribute), 67
- KEY (*pandagg.query.Script* attribute), 69
- KEY (*pandagg.query.ScriptScore* attribute), 69
- KEY (*pandagg.query.Shape* attribute), 68
- KEY (*pandagg.query.SimpleQueryString* attribute), 68
- KEY (*pandagg.query.Term* attribute), 67
- KEY (*pandagg.query.Terms* attribute), 67
- KEY (*pandagg.query.TermsSet* attribute), 67
- KEY (*pandagg.query.Type* attribute), 67
- KEY (*pandagg.query.Wildcard* attribute), 67
- KEY (*pandagg.query.Wrapper* attribute), 69
- KEY (*pandagg.tree.aggs.aggs.AbstractLeafAgg* attribute), 37
- KEY (*pandagg.tree.aggs.aggs.AbstractParentAgg* attribute), 37
- KEY (*pandagg.tree.aggs.bucket.Composite* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.DateHistogram* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.DateRange* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.Filter* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.Filters* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.Global* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.Histogram* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.Missing* attribute), 41
- KEY (*pandagg.tree.aggs.bucket.Nested* attribute), 42
- KEY (*pandagg.tree.aggs.bucket.Range* attribute), 42
- KEY (*pandagg.tree.aggs.bucket.ReverseNested* attribute), 42
- KEY (*pandagg.tree.aggs.bucket.Terms* attribute), 42
- KEY (*pandagg.tree.aggs.metric.Avg* attribute), 42
- KEY (*pandagg.tree.aggs.metric.Cardinality* attribute), 42
- KEY (*pandagg.tree.aggs.metric.ExtendedStats* attribute), 42
- KEY (*pandagg.tree.aggs.metric.GeoBound* attribute), 42

- KEY (*pandagg.tree.aggs.metric.GeoCentroid* attribute), 42
- KEY (*pandagg.tree.aggs.metric.Max* attribute), 42
- KEY (*pandagg.tree.aggs.metric.Min* attribute), 42
- KEY (*pandagg.tree.aggs.metric.PercentileRanks* attribute), 42
- KEY (*pandagg.tree.aggs.metric.Percentiles* attribute), 43
- KEY (*pandagg.tree.aggs.metric.Stats* attribute), 43
- KEY (*pandagg.tree.aggs.metric.Sum* attribute), 43
- KEY (*pandagg.tree.aggs.metric.TopHits* attribute), 43
- KEY (*pandagg.tree.aggs.metric.ValueCount* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.AvgBucket* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.BucketScript* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.BucketSelector* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.BucketSort* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.CumulativeSum* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.Derivative* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.ExtendedStatsBucket* attribute), 43
- KEY (*pandagg.tree.aggs.pipeline.MaxBucket* attribute), 44
- KEY (*pandagg.tree.aggs.pipeline.MinBucket* attribute), 44
- KEY (*pandagg.tree.aggs.pipeline.MovingAvg* attribute), 44
- KEY (*pandagg.tree.aggs.pipeline.PercentilesBucket* attribute), 44
- KEY (*pandagg.tree.aggs.pipeline.SerialDiff* attribute), 44
- KEY (*pandagg.tree.aggs.pipeline.StatsBucket* attribute), 44
- KEY (*pandagg.tree.aggs.pipeline.SumBucket* attribute), 44
- KEY (*pandagg.tree.mapping.Mapping* attribute), 50
- KEY (*pandagg.tree.query.abstract.Compound* attribute), 44
- KEY (*pandagg.tree.query.abstract.Leaf* attribute), 44
- KEY (*pandagg.tree.query.abstract.Query* attribute), 45
- KEY (*pandagg.tree.query.compound.Bool* attribute), 47
- KEY (*pandagg.tree.query.compound.Boosting* attribute), 47
- KEY (*pandagg.tree.query.compound.ConstantScore* attribute), 47
- KEY (*pandagg.tree.query.compound.DisMax* attribute), 47
- KEY (*pandagg.tree.query.compound.FunctionScore* attribute), 47
- KEY (*pandagg.tree.query.full_text.Common* attribute), 47
- KEY (*pandagg.tree.query.full_text.Intervals* attribute), 47
- KEY (*pandagg.tree.query.full_text.Match* attribute), 47
- KEY (*pandagg.tree.query.full_text.MatchBoolPrefix* attribute), 47
- KEY (*pandagg.tree.query.full_text.MatchPhrase* attribute), 47
- KEY (*pandagg.tree.query.full_text.MatchPhrasePrefix* attribute), 47
- KEY (*pandagg.tree.query.full_text.MultiMatch* attribute), 47
- KEY (*pandagg.tree.query.full_text.QueryString* attribute), 48
- KEY (*pandagg.tree.query.full_text.SimpleQueryString* attribute), 48
- KEY (*pandagg.tree.query.geo.GeoBoundingBox* attribute), 48
- KEY (*pandagg.tree.query.geo.GeoDistance* attribute), 48
- KEY (*pandagg.tree.query.geo.GeoPolygone* attribute), 48
- KEY (*pandagg.tree.query.geo.GeoShape* attribute), 48
- KEY (*pandagg.tree.query.joining.HasChild* attribute), 48
- KEY (*pandagg.tree.query.joining.HasParent* attribute), 48
- KEY (*pandagg.tree.query.joining.Nested* attribute), 48
- KEY (*pandagg.tree.query.joining.ParentId* attribute), 48
- KEY (*pandagg.tree.query.shape.Shape* attribute), 48
- KEY (*pandagg.tree.query.specialized.DistanceFeature* attribute), 49
- KEY (*pandagg.tree.query.specialized.MoreLikeThis* attribute), 49
- KEY (*pandagg.tree.query.specialized.Percolate* attribute), 49
- KEY (*pandagg.tree.query.specialized.RankFeature* attribute), 49
- KEY (*pandagg.tree.query.specialized.Script* attribute), 49
- KEY (*pandagg.tree.query.specialized.Wrapper* attribute), 49
- KEY (*pandagg.tree.query.specialized_compound.PinnedQuery* attribute), 49
- KEY (*pandagg.tree.query.specialized_compound.ScriptScore* attribute), 49
- KEY (*pandagg.tree.query.term_level.Exists* attribute), 49
- KEY (*pandagg.tree.query.term_level.Fuzzy* attribute), 49
- KEY (*pandagg.tree.query.term_level.Ids* attribute), 49
- KEY (*pandagg.tree.query.term_level.Prefix* attribute), 50
- KEY (*pandagg.tree.query.term_level.Range* attribute), 50
- KEY (*pandagg.tree.query.term_level.Regexp* attribute), 50
- KEY (*pandagg.tree.query.term_level.Term* attribute), 50
- KEY (*pandagg.tree.query.term_level.Terms* attribute), 50
- KEY (*pandagg.tree.query.term_level.TermsSet* attribute), 50
- KEY (*pandagg.tree.query.term_level.Type* attribute), 50
- KEY (*pandagg.tree.query.term_level.Wildcard* attribute), 50
- KEY_SEP (*pandagg.node.aggs.bucket.DateRange* attribute), 21
- KEY_SEP (*pandagg.node.aggs.bucket.Range* attribute),

22
 KeyFieldQueryClause (class in pandagg.node.query.abstract), 32
 keys() (pandagg.response.Aggregations method), 70
 Keyword (class in pandagg.mapping), 60
 Keyword (class in pandagg.node.mapping.field_datatypes), 29
 29
L
 Leaf (class in pandagg.tree.query.abstract), 44
 LeafQueryClause (class in pandagg.node.query.abstract), 32
 line_repr() (pandagg.node.aggs.abstract.AggregNode method), 18
 line_repr() (pandagg.node.aggs.abstract.ShadowRoot method), 20
 line_repr() (pandagg.node.mapping.abstract.Field method), 26
 line_repr() (pandagg.node.query.abstract.KeyFieldQueryClause method), 32
 line_repr() (pandagg.node.query.abstract.MultiFieldsQueryClause method), 32
 line_repr() (pandagg.node.query.abstract.QueryClause method), 32
 line_repr() (pandagg.node.query.geo.GeoDistance method), 34
 line_repr() (pandagg.node.query.term_level.Exists method), 36
 line_repr() (pandagg.node.query.term_level.Ids method), 36
 line_repr() (pandagg.node.response.bucket.Bucket method), 37
 list_documents() (pandagg.interactive.response.IResponse method), 17
 list_nesteds_at_field() (pandagg.mapping.Mapping method), 59
 list_nesteds_at_field() (pandagg.tree.mapping.Mapping method), 50
 Long (class in pandagg.mapping), 60
 Long (class in pandagg.node.mapping.field_datatypes), 29
 LongRange (class in pandagg.mapping), 61
 LongRange (class in pandagg.node.mapping.field_datatypes), 29
M
 MapperAnnotatedText (class in pandagg.mapping), 62
 MapperAnnotatedText (class in pandagg.node.mapping.field_datatypes), 29
 MapperMurMur3 (class in pandagg.mapping), 62
 MapperMurMur3 (class in pandagg.node.mapping.field_datatypes), 29
 Mapping (class in pandagg.mapping), 59
 Mapping (class in pandagg.tree.mapping), 50
 mapping_type_of_field() (pandagg.mapping.Mapping method), 59
 mapping_type_of_field() (pandagg.tree.mapping.Mapping method), 50
 MappingError, 59
 Match (class in pandagg.node.query.full_text), 33
 Match (class in pandagg.query), 67
 Match (class in pandagg.tree.query.full_text), 47
 MatchAll (class in pandagg.node.aggs.bucket), 21
 MatchBoolPrefix (class in pandagg.node.query.full_text), 33
 MatchBoolPrefix (class in pandagg.query), 67
 MatchBoolPrefix (class in pandagg.tree.query.full_text), 47
 MatchClause (class in pandagg.node.query.full_text), 33
 MatchPhrase (class in pandagg.query), 67
 MatchPhrase (class in pandagg.tree.query.full_text), 47
 MatchPhrasePrefix (class in pandagg.node.query.full_text), 33
 MatchPhrasePrefix (class in pandagg.query), 67
 MatchPhrasePrefix (class in pandagg.tree.query.full_text), 47
 Max (class in pandagg.aggs), 56
 Max (class in pandagg.node.aggs.metric), 23
 Max (class in pandagg.tree.aggs.metric), 42
 MaxBucket (class in pandagg.aggs), 57
 MaxBucket (class in pandagg.node.aggs.pipeline), 25
 MaxBucket (class in pandagg.tree.aggs.pipeline), 44
 Meta (class in pandagg.mapping), 64
 Meta (class in pandagg.node.mapping.meta_fields), 31
 MetricAgg (class in pandagg.node.aggs.abstract), 19
 Min (class in pandagg.aggs), 56
 Min (class in pandagg.node.aggs.metric), 23
 Min (class in pandagg.tree.aggs.metric), 42
 MinBucket (class in pandagg.aggs), 57
 MinBucket (class in pandagg.node.aggs.pipeline), 25
 MinBucket (class in pandagg.tree.aggs.pipeline), 44
 Missing (class in pandagg.aggs), 56
 Missing (class in pandagg.node.aggs.bucket), 21
 Missing (class in pandagg.tree.aggs.bucket), 41
 MoreLikeThis (class in pandagg.node.query.specialized), 35
 MoreLikeThis (class in pandagg.query), 69
 MoreLikeThis (class in pandagg.tree.query.specialized), 49
 MovingAvg (class in pandagg.aggs), 57

- MovingAvg (class in pandagg.node.aggs.pipeline), 25
- MovingAvg (class in pandagg.tree.aggs.pipeline), 44
- MultiFieldsQueryClause (class in pandagg.node.query.abstract), 32
- MultiMatch (class in pandagg.node.query.full_text), 33
- MultiMatch (class in pandagg.query), 67
- MultiMatch (class in pandagg.tree.query.full_text), 47
- MULTIPLE (pandagg.node.query.abstract.ParentParameterClause attribute), 32
- MultipleBucketAgg (class in pandagg.node.aggs.abstract), 19
- MultiSearch (class in pandagg.search), 71
- must () (pandagg.query.Query method), 65
- must () (pandagg.search.Search method), 75
- must () (pandagg.tree.query.abstract.Query method), 45
- must_not () (pandagg.query.Query method), 65
- must_not () (pandagg.search.Search method), 75
- must_not () (pandagg.tree.query.abstract.Query method), 45
- ## N
- name (pandagg.node.query.abstract.QueryClause attribute), 32
- Nested (class in pandagg.aggs), 56
- Nested (class in pandagg.mapping), 61
- Nested (class in pandagg.node.aggs.bucket), 22
- Nested (class in pandagg.node.mapping.field_datatypes), 29
- Nested (class in pandagg.node.query.joining), 34
- Nested (class in pandagg.query), 68
- Nested (class in pandagg.tree.aggs.bucket), 41
- Nested (class in pandagg.tree.query.joining), 48
- nested () (pandagg.query.Query method), 65
- nested () (pandagg.tree.query.abstract.Query method), 45
- nested_at_field () (pandagg.mapping.Mapping method), 60
- nested_at_field () (pandagg.tree.mapping.Mapping method), 50
- node_class (pandagg.aggs.Aggs attribute), 55
- node_class (pandagg.mapping.Mapping attribute), 60
- node_class (pandagg.query.Query attribute), 65
- node_class (pandagg.tree.aggs.aggs.Aggs attribute), 40
- node_class (pandagg.tree.mapping.Mapping attribute), 50
- node_class (pandagg.tree.query.abstract.Query attribute), 45
- node_path () (pandagg.mapping.Mapping method), 60
- node_path () (pandagg.tree.mapping.Mapping method), 51
- ## O
- Object (class in pandagg.mapping), 61
- Object (class in pandagg.node.mapping.field_datatypes), 29
- operator () (pandagg.node.query.compound.CompoundClause class method), 33
- ordered () (in module pandagg.utils), 77
- ## P
- pandagg (module), 78
- pandagg.aggs (module), 52
- pandagg.connections (module), 58
- pandagg.discovery (module), 59
- pandagg.exceptions (module), 59
- pandagg.interactive (module), 18
- pandagg.interactive.mapping (module), 17
- pandagg.interactive.response (module), 17
- pandagg.mapping (module), 59
- pandagg.node (module), 37
- pandagg.node.aggs (module), 26
- pandagg.node.aggs.abstract (module), 18
- pandagg.node.aggs.bucket (module), 20
- pandagg.node.aggs.metric (module), 23
- pandagg.node.aggs.pipeline (module), 24
- pandagg.node.mapping (module), 31
- pandagg.node.mapping.abstract (module), 26
- pandagg.node.mapping.field_datatypes (module), 27
- pandagg.node.mapping.meta_fields (module), 30
- pandagg.node.query (module), 37
- pandagg.node.query.abstract (module), 31
- pandagg.node.query.compound (module), 32
- pandagg.node.query.full_text (module), 33
- pandagg.node.query.geo (module), 34
- pandagg.node.query.joining (module), 34
- pandagg.node.query.shape (module), 35
- pandagg.node.query.span (module), 35
- pandagg.node.query.specialized (module), 35
- pandagg.node.query.specialized_compound (module), 35
- pandagg.node.query.term_level (module), 35
- pandagg.node.response (module), 37
- pandagg.node.response.bucket (module), 37
- pandagg.node.types (module), 37
- pandagg.query (module), 64
- pandagg.response (module), 69
- pandagg.search (module), 71
- pandagg.tree (module), 52
- pandagg.tree.aggs (module), 44

[pandagg.tree.aggs.aggs \(module\)](#), 37
[pandagg.tree.aggs.bucket \(module\)](#), 41
[pandagg.tree.aggs.metric \(module\)](#), 42
[pandagg.tree.aggs.pipeline \(module\)](#), 43
[pandagg.tree.mapping \(module\)](#), 50
[pandagg.tree.query \(module\)](#), 50
[pandagg.tree.query.abstract \(module\)](#), 44
[pandagg.tree.query.compound \(module\)](#), 47
[pandagg.tree.query.full_text \(module\)](#), 47
[pandagg.tree.query.geo \(module\)](#), 48
[pandagg.tree.query.joining \(module\)](#), 48
[pandagg.tree.query.shape \(module\)](#), 48
[pandagg.tree.query.span \(module\)](#), 49
[pandagg.tree.query.specialized \(module\)](#), 49
[pandagg.tree.query.specialized_compound \(module\)](#), 49
[pandagg.tree.query.term_level \(module\)](#), 49
[pandagg.tree.response \(module\)](#), 51
[pandagg.utils \(module\)](#), 77
[params \(\) \(pandagg.search.Request method\)](#), 71
[parent_id \(\) \(pandagg.query.Query method\)](#), 65
[parent_id \(\) \(pandagg.tree.query.abstract.Query method\)](#), 45
[ParentId \(class in pandagg.node.query.joining\)](#), 34
[ParentId \(class in pandagg.query\)](#), 68
[ParentId \(class in pandagg.tree.query.joining\)](#), 48
[ParentParameterClause \(class in pandagg.node.query.abstract\)](#), 32
[parse \(\) \(pandagg.tree.response.AggsResponseTree method\)](#), 51
[PercentileRanks \(class in pandagg.aggs\)](#), 57
[PercentileRanks \(class in pandagg.node.aggs.metric\)](#), 23
[PercentileRanks \(class in pandagg.tree.aggs.metric\)](#), 42
[Percentiles \(class in pandagg.aggs\)](#), 56
[Percentiles \(class in pandagg.node.aggs.metric\)](#), 24
[Percentiles \(class in pandagg.tree.aggs.metric\)](#), 42
[PercentilesBucket \(class in pandagg.aggs\)](#), 57
[PercentilesBucket \(class in pandagg.node.aggs.pipeline\)](#), 25
[PercentilesBucket \(class in pandagg.tree.aggs.pipeline\)](#), 44
[Percolate \(class in pandagg.node.query.specialized\)](#), 35
[Percolate \(class in pandagg.query\)](#), 69
[Percolate \(class in pandagg.tree.query.specialized\)](#), 49
[Percolator \(class in pandagg.mapping\)](#), 62
[Percolator \(class in pandagg.node.mapping.field_datatypes\)](#), 29
[pinned_query \(\) \(pandagg.query.Query method\)](#), 65

[pinned_query \(\) \(pandagg.tree.query.abstract.Query method\)](#), 45
[PinnedQuery \(class in pandagg.node.query.specialized_compound\)](#), 35
[PinnedQuery \(class in pandagg.query\)](#), 69
[PinnedQuery \(class in pandagg.tree.query.specialized_compound\)](#), 49
[Pipeline \(class in pandagg.node.aggs.abstract\)](#), 19
[post_filter \(\) \(pandagg.search.Search method\)](#), 75
[Prefix \(class in pandagg.node.query.term_level\)](#), 36
[Prefix \(class in pandagg.query\)](#), 66
[Prefix \(class in pandagg.tree.query.term_level\)](#), 50

Q

[Query \(class in pandagg.query\)](#), 64
[Query \(class in pandagg.tree.query.abstract\)](#), 44
[query \(\) \(pandagg.query.Query method\)](#), 65
[query \(\) \(pandagg.search.Search method\)](#), 75
[query \(\) \(pandagg.tree.query.abstract.Query method\)](#), 45
[QueryClause \(class in pandagg.node.query.abstract\)](#), 32
[QueryString \(class in pandagg.node.query.full_text\)](#), 34
[QueryString \(class in pandagg.query\)](#), 68
[QueryString \(class in pandagg.tree.query.full_text\)](#), 47

R

[Range \(class in pandagg.aggs\)](#), 55
[Range \(class in pandagg.node.aggs.bucket\)](#), 22
[Range \(class in pandagg.node.query.term_level\)](#), 36
[Range \(class in pandagg.query\)](#), 67
[Range \(class in pandagg.tree.aggs.bucket\)](#), 42
[Range \(class in pandagg.tree.query.term_level\)](#), 50
[RankFeature \(class in pandagg.mapping\)](#), 62
[RankFeature \(class in pandagg.node.mapping.field_datatypes\)](#), 29
[RankFeature \(class in pandagg.node.query.specialized\)](#), 35
[RankFeature \(class in pandagg.query\)](#), 69
[RankFeature \(class in pandagg.tree.query.specialized\)](#), 49
[RankFeatures \(class in pandagg.mapping\)](#), 62
[RankFeatures \(class in pandagg.node.mapping.field_datatypes\)](#), 29
[Regex \(class in pandagg.node.query.term_level\)](#), 36
[Regex \(class in pandagg.query\)](#), 67
[Regex \(class in pandagg.tree.query.term_level\)](#), 50
[remove_connection \(\) \(pandagg.connections.Connections method\)](#),

- 58
- Request (class in pandagg.search), 71
- resolve_path_to_id() (pandagg.mapping.Mapping method), 60
- resolve_path_to_id() (pandagg.tree.mapping.Mapping method), 51
- Response (class in pandagg.response), 70
- ReverseNested (class in pandagg.aggs), 56
- ReverseNested (class in pandagg.node.aggs.bucket), 22
- ReverseNested (class in pandagg.tree.aggs.bucket), 42
- ROOT_NAME (pandagg.node.response.bucket.Bucket attribute), 37
- Routing (class in pandagg.mapping), 64
- Routing (class in pandagg.node.mapping.meta_fields), 31
- ## S
- ScaledFloat (class in pandagg.mapping), 60
- ScaledFloat (class in pandagg.node.mapping.field_datatypes), 29
- scan() (pandagg.search.Search method), 75
- Script (class in pandagg.node.query.specialized), 35
- Script (class in pandagg.query), 69
- Script (class in pandagg.tree.query.specialized), 49
- script_fields() (pandagg.search.Search method), 76
- script_score() (pandagg.query.Query method), 66
- script_score() (pandagg.tree.query.abstract.Query method), 46
- ScriptPipeline (class in pandagg.node.aggs.abstract), 20
- ScriptScore (class in pandagg.node.query.specialized_compound), 35
- ScriptScore (class in pandagg.query), 69
- ScriptScore (class in pandagg.tree.query.specialized_compound), 49
- Search (class in pandagg.search), 71
- search() (pandagg.discovery.Index method), 59
- SearchAsYouType (class in pandagg.mapping), 63
- SearchAsYouType (class in pandagg.node.mapping.field_datatypes), 30
- SerialDiff (class in pandagg.aggs), 58
- SerialDiff (class in pandagg.node.aggs.pipeline), 26
- SerialDiff (class in pandagg.tree.aggs.pipeline), 44
- serialize() (pandagg.response.Aggregations method), 70
- ShadowRoot (class in pandagg.node.aggs.abstract), 20
- ShadowRoot (class in pandagg.node.mapping.abstract), 26
- Shape (class in pandagg.mapping), 63
- Shape (class in pandagg.node.mapping.field_datatypes), 30
- Shape (class in pandagg.node.query.shape), 35
- Shape (class in pandagg.query), 68
- Shape (class in pandagg.tree.query.shape), 48
- Short (class in pandagg.mapping), 60
- Short (class in pandagg.node.mapping.field_datatypes), 30
- should() (pandagg.query.Query method), 66
- should() (pandagg.search.Search method), 76
- should() (pandagg.tree.query.abstract.Query method), 46
- show() (pandagg.aggs.Aggs method), 55
- show() (pandagg.query.Query method), 66
- show() (pandagg.tree.aggs.aggs.Aggs method), 40
- show() (pandagg.tree.query.abstract.Query method), 46
- show() (pandagg.tree.response.AggsResponseTree method), 51
- SimpleQueryString (class in pandagg.node.query.full_text), 34
- SimpleQueryString (class in pandagg.query), 68
- SimpleQueryString (class in pandagg.tree.query.full_text), 48
- Size (class in pandagg.mapping), 64
- Size (class in pandagg.node.mapping.meta_fields), 31
- size() (pandagg.search.Search method), 76
- sort() (pandagg.search.Search method), 76
- Source (class in pandagg.mapping), 64
- Source (class in pandagg.node.mapping.meta_fields), 31
- source() (pandagg.search.Search method), 76
- SparseVector (class in pandagg.mapping), 63
- SparseVector (class in pandagg.node.mapping.field_datatypes), 30
- Stats (class in pandagg.aggs), 56
- Stats (class in pandagg.node.aggs.metric), 24
- Stats (class in pandagg.tree.aggs.metric), 43
- StatsBucket (class in pandagg.aggs), 57
- StatsBucket (class in pandagg.node.aggs.pipeline), 26
- StatsBucket (class in pandagg.tree.aggs.pipeline), 44
- success (pandagg.response.Response attribute), 70
- suggest() (pandagg.search.Search method), 77
- Sum (class in pandagg.aggs), 56
- Sum (class in pandagg.node.aggs.metric), 24
- Sum (class in pandagg.tree.aggs.metric), 43
- SumBucket (class in pandagg.aggs), 57
- SumBucket (class in pandagg.node.aggs.pipeline), 26
- SumBucket (class in pandagg.tree.aggs.pipeline), 44

T

Term (class in *pandagg.node.query.term_level*), 36
 Term (class in *pandagg.query*), 67
 Term (class in *pandagg.tree.query.term_level*), 50
 Terms (class in *pandagg.aggs*), 55
 Terms (class in *pandagg.node.aggs.bucket*), 22
 Terms (class in *pandagg.node.query.term_level*), 36
 Terms (class in *pandagg.query*), 67
 Terms (class in *pandagg.tree.aggs.bucket*), 42
 Terms (class in *pandagg.tree.query.term_level*), 50
 TermsSet (class in *pandagg.node.query.term_level*), 36
 TermsSet (class in *pandagg.query*), 67
 TermsSet (class in *pandagg.tree.query.term_level*), 50
 Text (class in *pandagg.mapping*), 60
 Text (class in *pandagg.node.mapping.field_datatypes*), 30
 to_dataframe() (*pandagg.response.Aggregations* method), 70
 to_dict() (*pandagg.aggs.Aggs* method), 55
 to_dict() (*pandagg.mapping.Mapping* method), 60
 to_dict() (*pandagg.node.aggs.abstract.AgNode* method), 18
 to_dict() (*pandagg.node.query.abstract.QueryClause* method), 32
 to_dict() (*pandagg.node.query.term_level.Ids* method), 36
 to_dict() (*pandagg.query.Query* method), 66
 to_dict() (*pandagg.search.MultiSearch* method), 71
 to_dict() (*pandagg.search.Search* method), 77
 to_dict() (*pandagg.tree.aggs.aggs.Aggs* method), 41
 to_dict() (*pandagg.tree.mapping.Mapping* method), 51
 to_dict() (*pandagg.tree.query.abstract.Query* method), 46
 to_interactive_tree() (*pandagg.response.Aggregations* method), 70
 to_key (*pandagg.node.aggs.bucket.Range* attribute), 22
 to_named_field() (*pandagg.node.mapping.abstract.UnnamedField* attribute), 27
 to_normalized() (*pandagg.response.Aggregations* method), 70
 to_tabular() (*pandagg.response.Aggregations* method), 70
 to_tree() (*pandagg.response.Aggregations* method), 70
 TokenCount (class in *pandagg.mapping*), 62
 TokenCount (class in *pandagg.node.mapping.field_datatypes*), 30
 TopHits (class in *pandagg.aggs*), 57
 TopHits (class in *pandagg.node.aggs.metric*), 24
 TopHits (class in *pandagg.tree.aggs.metric*), 43
 Type (class in *pandagg.mapping*), 63

Type (class in *pandagg.node.mapping.meta_fields*), 31
 Type (class in *pandagg.node.query.term_level*), 36
 Type (class in *pandagg.query*), 67
 Type (class in *pandagg.tree.query.term_level*), 50

U

UniqueBucketAgg (class in *pandagg.node.aggs.abstract*), 20
 UnnamedComplexField (class in *pandagg.node.mapping.abstract*), 26
 UnnamedField (class in *pandagg.node.mapping.abstract*), 26
 UnnamedRegularField (class in *pandagg.node.mapping.abstract*), 27
 update_from_dict() (*pandagg.search.Search* method), 77
 using() (*pandagg.search.Request* method), 71

V

valid_on_field_type() (*pandagg.node.aggs.abstract.AgNode* class method), 19
 validate_agg_node() (*pandagg.mapping.Mapping* method), 60
 validate_agg_node() (*pandagg.tree.mapping.Mapping* method), 51
 VALUE_ATTRS (*pandagg.node.aggs.abstract.AgNode* attribute), 18
 VALUE_ATTRS (*pandagg.node.aggs.abstract.BucketAggNode* attribute), 19
 VALUE_ATTRS (*pandagg.node.aggs.abstract.FieldOrScriptMetricAgg* attribute), 19
 VALUE_ATTRS (*pandagg.node.aggs.abstract.MetricAgg* attribute), 19
 VALUE_ATTRS (*pandagg.node.aggs.abstract.MultipleBucketAgg* attribute), 19
 VALUE_ATTRS (*pandagg.node.aggs.abstract.Pipeline* attribute), 19
 VALUE_ATTRS (*pandagg.node.aggs.abstract.ScriptPipeline* attribute), 20
 VALUE_ATTRS (*pandagg.node.aggs.abstract.UniqueBucketAgg* attribute), 20
 VALUE_ATTRS (*pandagg.node.aggs.bucket.DateHistogram* attribute), 20
 VALUE_ATTRS (*pandagg.node.aggs.bucket.DateRange* attribute), 21
 VALUE_ATTRS (*pandagg.node.aggs.bucket.Filter* attribute), 21
 VALUE_ATTRS (*pandagg.node.aggs.bucket.Filters* attribute), 21
 VALUE_ATTRS (*pandagg.node.aggs.bucket.Global* attribute), 21

- VALUE_ATTRS (*pandagg.node.aggs.bucket.Histogram attribute*), 21
- VALUE_ATTRS (*pandagg.node.aggs.bucket.Missing attribute*), 22
- VALUE_ATTRS (*pandagg.node.aggs.bucket.Nested attribute*), 22
- VALUE_ATTRS (*pandagg.node.aggs.bucket.Range attribute*), 22
- VALUE_ATTRS (*pandagg.node.aggs.bucket.ReverseNested attribute*), 22
- VALUE_ATTRS (*pandagg.node.aggs.bucket.Terms attribute*), 22
- VALUE_ATTRS (*pandagg.node.aggs.metric.Avg attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.Cardinality attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.ExtendedStats attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.GeoBound attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.GeoCentroid attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.Max attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.Min attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.PercentileRanks attribute*), 23
- VALUE_ATTRS (*pandagg.node.aggs.metric.Percentiles attribute*), 24
- VALUE_ATTRS (*pandagg.node.aggs.metric.Stats attribute*), 24
- VALUE_ATTRS (*pandagg.node.aggs.metric.Sum attribute*), 24
- VALUE_ATTRS (*pandagg.node.aggs.metric.TopHits attribute*), 24
- VALUE_ATTRS (*pandagg.node.aggs.metric.ValueCount attribute*), 24
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.AvgBucket attribute*), 24
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.BucketScript attribute*), 24
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.BucketSelector attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.BucketSort attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.CumulativeSum attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.Derivative attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.ExtendedStatsBucket attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.MaxBucket attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.MinBucket attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.MovingAvg attribute*), 25
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.PercentilesBucket attribute*), 26
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.SerialDiff attribute*), 26
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.StatsBucket attribute*), 26
- VALUE_ATTRS (*pandagg.node.aggs.pipeline.SumBucket attribute*), 26
- ValueCount (class in *pandagg.aggs*), 57
- ValueCount (class in *pandagg.node.aggs.metric*), 24
- ValueCount (class in *pandagg.tree.aggs.metric*), 43
- VersionIncompatibilityError, 59
- ## W
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.abstract.AggNode attribute*), 18
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.DateHistogram attribute*), 20
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.DateRange attribute*), 21
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.Histogram attribute*), 21
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.Nested attribute*), 22
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.Range attribute*), 22
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.bucket.ReverseNested attribute*), 22
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.metric.Avg attribute*), 23
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.metric.ExtendedStats attribute*), 23
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.metric.GeoBound attribute*), 23
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.metric.GeoCentroid attribute*), 23
- WHITELISTED_MAPPING_TYPES (*pandagg.node.aggs.metric.Max attribute*), 23

WHITELISTED_MAPPING_TYPES
(*pandagg.node.aggs.metric.Min* *attribute*),
[23](#)

WHITELISTED_MAPPING_TYPES
(*pandagg.node.aggs.metric.PercentileRanks*
attribute), [24](#)

WHITELISTED_MAPPING_TYPES
(*pandagg.node.aggs.metric.Percentiles* *at-*
tribute), [24](#)

WHITELISTED_MAPPING_TYPES
(*pandagg.node.aggs.metric.Stats* *attribute*),
[24](#)

WHITELISTED_MAPPING_TYPES
(*pandagg.node.aggs.metric.Sum* *attribute*),
[24](#)

Wildcard (*class in pandagg.node.query.term_level*), [36](#)

Wildcard (*class in pandagg.query*), [67](#)

Wildcard (*class in pandagg.tree.query.term_level*), [50](#)

Wrapper (*class in pandagg.node.query.specialized*), [35](#)

Wrapper (*class in pandagg.query*), [69](#)

Wrapper (*class in pandagg.tree.query.specialized*), [49](#)