pandasticsearch Documentation

Release 0.2.0

onesuper

Jul 27, 2021

Contents

Contents:

CHAPTER 1

pandasticsearch package

1.1 Submodules

1.2 pandasticsearch.client module

class pandasticsearch.client.**RestClient**(*host*, *username=None*, *password=None*, *verify_ssl=True*)

Bases: object

RestClient talks to Elasticsearch cluster through native RESTful API.

get(*path*, *params=None*)

Sends a GET request to Elasticsearch.

Parameters

- **path** – path: path of the verb and resource, e.g. /index
- **params** (*optional*) – Dictionary to be sent in the query string.

Returns The response as a dictionary.

```
>>> from pandasticsearch import RestClient
>>> client = RestClient('http://localhost:9200')
>>> print(client.get())
```

post(*path*, *data*, *params=None*)

Sends a POST request to Elasticsearch.

Parameters

- **path** – The path for the verb and resource
- **data** – The json data to send in the body of the request.
- **params** (*optional*) – Dictionary to be sent in the query string.

Returns The response as a dictionary.

```
>>> from pandasticsearch import RestClient
>>> client = RestClient('http://localhost:9200')
>>> print(client.post(path='/index/_search', data={"query":{"match_all":{}}}))
```

1.3 pandasticsearch.dataframe module

class pandasticsearch.dataframe.**DataFrame**(***kwargs*)

Bases: object

A *DataFrame* treats index and documents in Elasticsearch as named columns and rows.

```
>>> from pandasticsearch import DataFrame
>>> df = DataFrame.from_es('http://localhost:9200', index='people')
```

Customizing the endpoint of the ElasticSearch:

```
>>> from pandasticsearch import DataFrame
>>> from pandasticsearch.client import RestClient
>>> df = DataFrame(client=RestClient('http://host:port',), index='people')
```

It can be converted to Pandas object for subsequent analysis:

```
>>> df.to_pandas()
```

agg(**aggs*)

Aggregate on the entire DataFrame without groups.

Parameters **aggs** – a list of Aggregator objects

```
>>> df[df['gender'] == 'male'].agg(df['age'].avg).collect()
[Row(avg(age)=12)]
```

collect()

Returns all the records as a list of Row.

Returns list of *Row*

```
>>> df.collect()
[Row(age=2, name='Alice'), Row(age=5, name='Bob')]
```

columns

Returns all column names as a list.

Returns column names as a list

```
>>> df.columns
['age', 'name']
```

count()

Returns a list of numbers indicating the count for each group

```
>>> df.groupby(df.gender).count()
[2, 1]
```

filter(*condition*)

Filters rows using a given condition.

where() is an alias for filter().

Parameters **condition** – BooleanFilter object or a string

```
>>> df.filter(df['age'] < 13).collect()
[Row(age=12,gender='female',name='Alice'), Row(age=11,gender='male',name='Bob
↪')]
```

static from_es(***kwargs*)

Creates an *DataFrame* object by providing the URL of ElasticSearch node and the name of the index.

Parameters

- **url** (*str*) – URL of the node connected to (default: 'http://localhost:9200')
- **index** (*str*) – The name of the index
- **doc_type** (*str*) – The type of the document
- **compat** (*str*) – The compatible ES version (an integer number)

Returns DataFrame object for accessing

Return type *DataFrame*

```
>>> from pandasticsearch import DataFrame
>>> df = DataFrame.from_es('http://localhost:9200', index='people')
```

groupby(**cols*)

Returns a new *DataFrame* object grouped by the specified column(s).

Parameters **cols** – A list of column names, *Column* or Grouper objects

index

Returns the index name.

Returns string as the name

```
>>> df.index
people/children
```

limit(*num*)

Limits the result count to the number specified.

orderby(**cols*)

Returns a new *DataFrame* object sorted by the specified column(s).

Parameters **cols** – A list of column names, *Column* or Sorter.

orderby() is an alias for sort().

```
>>> df.sort(df['age'].asc).collect()
[Row(age=11,name='Bob'), Row(age=12,name='Alice'), Row(age=13,name='Leo')]
```

print_debug()

Post the query to the Elasticsearch Server and prints out the result it returned

print_schema()

Prints out the schema in the tree format.

```
>>> df.print_schema()
index_name
|-- type_name
```

(continues on next page)

(continued from previous page)

```
 |-- experience :  {'type': 'integer'}
 |-- id :  {'type': 'string'}
 |-- mobile :  {'index': 'not_analyzed', 'type': 'string'}
 |-- regions :  {'index': 'not_analyzed', 'type': 'string'}
```

classmethod resolve_mappings(*json_map*)

resolve_schema(*json_prop*, *res_schema=''*, *depth=1*)

schema
Returns the schema(mapping) of the index/type as a dictionary.

select(**cols*)
Projects a set of columns and returns a new *DataFrame*

Parameters **cols** – list of column names or *Column*.

```
>>> df.filter(df['age'] < 25).select('name', 'age').collect()
[Row(age=12,name='Alice'), Row(age=11,name='Bob'), Row(age=13,name='Leo')]
```

show(*n=200*, *truncate=15*)
Prints the first n rows to the console.

Parameters

- **n** – Number of rows to show.
- **truncate** – Number of words to be truncated for each column.

```
>>> df.filter(df['age'] < 25).select('name').show(3)
+------+
| name |
+------+
| Alice|
| Bob  |
| Leo  |
+------+
```

sort(**cols*)
Returns a new *DataFrame* object sorted by the specified column(s).

Parameters **cols** – A list of column names, *Column* or Sorter.

orderby() is an alias for sort().

```
>>> df.sort(df['age'].asc).collect()
[Row(age=11,name='Bob'), Row(age=12,name='Alice'), Row(age=13,name='Leo')]
```

to_dict()
Converts the current *DataFrame* object to Elasticsearch search dictionary.

Returns a dictionary which obeys the Elasticsearch RESTful protocol

to_pandas()
Export to a Pandas DataFrame object.

Returns The DataFrame representing the query result

```
>>> df[df['gender'] == 'male'].agg(Avg('age')).to_pandas()
    avg(age)
0         12
```

where(*condition*)
Filters rows using a given condition.

where() is an alias for filter().

Parameters **condition** – BooleanFilter object or a string

```
>>> df.filter(df['age'] < 13).collect()
[Row(age=12,gender='female',name='Alice'), Row(age=11,gender='male',name='Bob
↪')]
```

1.4 pandasticsearch.errors module

exception pandasticsearch.errors.**DataFrameException**(*msg*)
Bases: *pandasticsearch.errors.PandasticSearchException*

exception pandasticsearch.errors.**NoSuchDependencyException**(*msg*)
Bases: *pandasticsearch.errors.PandasticSearchException*

exception pandasticsearch.errors.**PandasticSearchException**(*msg*)
Bases: exceptions.RuntimeError

exception pandasticsearch.errors.**ParseResultException**(*msg*)
Bases: *pandasticsearch.errors.PandasticSearchException*

exception pandasticsearch.errors.**ServerDefinedException**(*msg*)
Bases: *pandasticsearch.errors.PandasticSearchException*

1.5 pandasticsearch.operators module

1.6 pandasticsearch.queries module

class pandasticsearch.queries.**Agg**
Bases: *pandasticsearch.queries.Query*

explain_result(*result=None*)

static from_dict(*d*)

index

to_pandas()
Export the current query result to a Pandas DataFrame object.

class pandasticsearch.queries.**Query**
Bases: _abcoll.MutableSequence

append(*value*)
S.append(object) – append object to the end of the sequence

explain_result(*result=None*)

insert(*index*, *value*)
S.insert(index, object) – insert object before index

json
Gets the original JSON representation returned by Elasticsearch REST API :return: The JSON string indicating the query result :rtype: string

millis_taken

print_json()

result

to_pandas()
Export the current query result to a Pandas DataFrame object.

class pandasticsearch.queries.**ScrollSelect**(*hits_generator*)
Bases: *pandasticsearch.queries.Select*

millis_taken/json not supported for ScrollSelect

result

row_generator()

to_pandas()
Export the current query result to a Pandas DataFrame object.

class pandasticsearch.queries.**Select**
Bases: *pandasticsearch.queries.Query*

explain_result(*result=None*)

static from_dict(*d*)

hit_to_row(*hit*)

resolve_fields(*row*)

result_as_tabular(*cols*, *n*, *truncate=20*)

to_pandas()
Export the current query result to a Pandas DataFrame object.

1.7 pandasticsearch.types module

class pandasticsearch.types.**Column**(*field*)
Bases: object

asc
Ascending Sorter

Returns Sorter

```
>>> df.orderyby(df.age.asc)
```

avg
Avg aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.avg)
```

cardinality
Distince aggregator

Returns `Aggregator`

```
>>> df.groupby(df.gender).agg(df.age.cardinality)
>>> df.groupby(df.gender).agg(df.age.distinct_count)
```

count
Value count aggregator

Returns `Aggregator`

```
>>> df.groupby(df.gender).agg(df.age.value_count)
```

date_interval(*interval*, *format='yyyy/MM/dd HH:mm:ss'*)
Returns a `Grouper`

Parameters

- **interval** – A string indicating date interval
- **format** – Date format string

Returns `Grouper`

```
>>> df.groupby(df.date_interval('1d'))
```

desc
Descending `Sorter`

Returns `Sorter`

```
>>> df.orderyby(df.age.desc)
```

distinct_count
Distince aggregator

Returns `Aggregator`

```
>>> df.groupby(df.gender).agg(df.age.cardinality)
>>> df.groupby(df.gender).agg(df.age.distinct_count)
```

extended_stats
Extended stats aggregator

Returns `Aggregator`

```
>>> df.groupby(df.gender).agg(df.age.extended_stats)
```

field_name()

isin(*values*)
Returns a `BooleanFilter`

Parameters **values** – A list of values to filter terms

Returns `BooleanFilter`

df.filter(df.gender.isin([‘male’, ‘female’])

isnull
`BooleanFilter` to indicate the null column value

Returns `BooleanFilter`

like (*wildcard*)
Returns a BooleanFilter

Parameters **wildcard** (*str*) – The wildcard to filter the column with.

Returns BooleanFilter

```
>>> df.filter(df.name.like('A*'))
```

max
Max aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.max)
```

min
Min aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.min)
```

notnull
BooleanFilter to indicate the non-null column value

Returns BooleanFilter

percentile_ranks
Percentile ranks aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.percentile_ranks)
```

percentiles
Percentile aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.percentiles)
```

ranges (*values*)
Returns a Grouper

Parameters **values** – A list of numeric values

Returns Grouper

```
>>> df.groupby(df.age.ranges([10,12,14]))
```

rlike (*regexp*)
Returns a BooleanFilter

Parameters **regexp** (*str*) – The regular expression to filter the column with.

Returns BooleanFilter

```
>>> df.filter(df.name.rlike('A.l.e'))
```

startswith (*substr*)
Returns a BooleanFilter

Parameters **substr** (*str*) – The sub string to filter the column with.

Returns BooleanFilter

```
>>> df.filter(df.name.startswith('Al')
```

stats
Stats aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.stats)
```

sum
Sum aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.sum)
```

terms (*limit=20*, *include=None*, *exclude=None*)
Returns a Grouper

Parameters

- **limit** – limit the number of terms to be aggregated (default 20)
- **include** – the exact term to be included
- **exclude** – the exact term to be excluded

Returns Grouper

```
>>> df.groupby(df.age.terms(limit=10, include=[1, 2, 3]))
```

value_count
Value count aggregator

Returns Aggregator

```
>>> df.groupby(df.gender).agg(df.age.value_count)
```

class pandasticsearch.types.**Row**
Bases: tuple

The builtin *DataFrame* row type for accessing before converted into Pandas DataFrame. The fields will be sorted by names.

```
>>> row = Row(name="Alice", age=12)
>>> row
Row(age=12, name='Alice')
>>> row['name'], row['age']
('Alice', 12)
>>> row.name, row.age
('Alice', 12)
>>> 'name' in row
True
>>> 'wrong_key' in row
```

as_dict()

1.8 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index