Package 'uptasticsearch'

July 22, 2025

Type Package

Title Get Data Frame Representations of 'Elasticsearch' Results

Version 1.0.0

Maintainer James Lamb <jaylamb20@gmail.com>

Description 'Elasticsearch' is an open-source, distributed, document-based datastore (<https://www.elastic.co/products/elasticsearch>).
It provides an 'HTTP' 'API' for querying the database and extracting datasets, but that 'API' was not designed for common data science workflows like pulling large batches of records and normalizing those documents into a data frame that can be used as a training dataset for statistical models. 'uptasticsearch' provides an interface for 'Elasticsearch' that is explicitly designed to make these data science workflows easy and fun.

Depends R (>= 3.3.0)

Imports curl, data.table, futile.logger, jsonlite, purrr, stats, stringr

Suggests knitr, markdown, testthat

License BSD_3_clause + file LICENSE

URL https://github.com/uptake/uptasticsearch

BugReports https://github.com/uptake/uptasticsearch/issues

RoxygenNote 7.3.2

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Author James Lamb [aut, cre],
Nick Paras [aut],
Austin Dickey [aut],
Michael Frasco [ctb],
Weiwen Gu [ctb],
Will Dearden [ctb],
Uptake Technologies Inc. [cph]

Repository CRAN

Date/Publication 2025-02-24 18:50:02 UTC

Contents

chomp_aggs 2
chomp_hits 3
es_search 4
get_fields 6
parse_date_time 7
unpack_nested_data 8

Index **9**

`chomp_aggs` *Aggs query to data.table*

Description

Given some raw JSON from an aggs query in Elasticsearch, parse the aggregations into a data.table.

Usage

```
chomp_aggs(aggs_json = NULL)
```

Arguments

`aggs_json`	A character vector. If its length is greater than 1, its elements will be pasted together. This can contain a JSON returned from an `aggs` query in Elasticsearch, or a filepath or URL pointing at one.

Value

A data.table representation of the result or NULL if the aggregation result is empty.

Examples

```
# A sample raw result from an aggs query combining date_histogram and extended_stats:
result <- '{"aggregations":{"dateTime":{"buckets":[{"key_as_string":"2016-12-01T00:00:00.000Z",
"key":1480550400000,"doc_count":123,"num_potatoes":{"count":120,"min":0,"max":40,"avg":15,
"sum":1800,"sum_of_squares":28000,"variance":225,"std_deviation":15,"std_deviation_bounds":{
"upper":26,"lower":13}}},{"key_as_string":"2017-01-01T00:00:00.000Z","key":1483228800000,
"doc_count":134,"num_potatoes":{"count":131,"min":0,"max":39,"avg":16,"sum":2096,
"sum_of_squares":34000,"variance":225,"std_deviation":15,"std_deviation_bounds":{"upper":26,
"lower":13}}}]}}}'

# Parse into a data.table
aggDT <- chomp_aggs(aggs_json = result)
print(aggDT)
```

chomp_hits *Hits to data.tables*

Description

A function for converting Elasticsearch docs into R data.tables. It uses fromJSON with flatten = TRUE to convert a JSON into an R data.frame, and formats it into a data.table.

Usage

```
chomp_hits(hits_json = NULL, keep_nested_data_cols = TRUE)
```

Arguments

hits_json	A character vector. If its length is greater than 1, its elements will be pasted together. This can contain a JSON returned from a search query in Elasticsearch, or a filepath or URL pointing at one.
keep_nested_data_cols	a boolean (default TRUE); whether to keep columns that are nested arrays in the original JSON. A warning will be given if these columns are deleted.

Examples

```
# A sample raw result from a hits query:
result <- '[{"_source":{"timestamp":"2017-01-01","cust_name":"Austin","details":{
"cust_class":"big_spender","location":"chicago","pastPurchases":[{"film":"The Notebook",
"pmt_amount":6.25},{"film":"The Town","pmt_amount":8.00},{"film":"Zootopia","pmt_amount":7.50,
"matinee":true}]}}},{"_source":{"timestamp":"2017-02-02","cust_name":"James","details":{
"cust_class":"peasant","location":"chicago","pastPurchases":[{"film":"Minions",
"pmt_amount":6.25,"matinee":true},{"film":"Rogue One","pmt_amount":10.25},{"film":"Bridesmaids",
"pmt_amount":8.75},{"film":"Bridesmaids","pmt_amount":6.25,"matinee":true}]}}},{"_source":{
"timestamp":"2017-03-03","cust_name":"Nick","details":{"cust_class":"critic","location":"cannes",
"pastPurchases":[{"film":"Aala Kaf Ifrit","pmt_amount":0,"matinee":true},{
"film":"Dopo la guerra (Apres la Guerre)","pmt_amount":0,"matinee":true},{
"film":"Avengers: Infinity War","pmt_amount":12.75}]}}}]'

# Chomp into a data.table
sampleChompedDT <- chomp_hits(hits_json = result, keep_nested_data_cols = TRUE)
print(sampleChompedDT)

# (Note: use es_search() to get here in one step)

# Unpack by details.pastPurchases
unpackedDT <- unpack_nested_data(chomped_df = sampleChompedDT
                                 , col_to_unpack = "details.pastPurchases")
print(unpackedDT)
```

es_search *Execute an Elasticsearch query and get a data.table*

Description

Given a query and some optional parameters, es_search gets results from HTTP requests to Elasticsearch and returns a data.table representation of those results.

Usage

```
es_search(
  es_host,
  es_index,
  size = 10000,
  query_body = "{}",
  scroll = "5m",
  max_hits = Inf,
  n_cores = ceiling(parallel::detectCores()/2),
  break_on_duplicates = TRUE,
  ignore_scroll_restriction = FALSE,
  intermediates_dir = getwd()
)
```

Arguments

es_host	A string identifying an Elasticsearch host. This should be of the form [transfer_protocol][hostname] For example, 'http://myindex.thing.com:9200'.
es_index	The name of an Elasticsearch index to be queried. Note that passing NULL is not supported. Technically, not passing an index to Elasticsearch is legal and results in searching over all indexes. To be sure that this very expensive query is not executed by accident, uptasticsearch forbids this. If you want to execute a query over all indexes in the cluster, set this argument to "_all".
size	Number of records per page of results. See Elasticsearch docs for more. Note that this will be reset to 0 if you submit a query_body with an "aggs" request in it. Also see max_hits.
query_body	String with a valid Elasticsearch query. Default is an empty query.
scroll	How long should the scroll context be held open? This should be a duration string like "1m" (for one minute) or "15s" (for 15 seconds). The scroll context will be refreshed every time you ask Elasticsearch for another record, so this parameter should just be the amount of time you expect to pass between requests. See the Elasticsearch scroll/pagination docs for more information.
max_hits	Integer. If specified, es_search will stop pulling data as soon as it has pulled this many hits. Default is Inf, meaning that all possible hits will be pulled.
n_cores	Number of cores to distribute fetching and processing over.

`break_on_duplicates`
: Boolean, defaults to TRUE. `es_search` uses the size of the final object it returns to check whether or not some data were lost during the processing. If you have duplicates in the source data, you will have to set this flag to FALSE and just trust that no data have been lost. Sorry :(.

`ignore_scroll_restriction`
: There is a cost associated with keeping an Elasticsearch scroll context open. By default, this function does not allow arguments to `scroll` which exceed one hour. This is done to prevent costly mistakes made by novice Elasticsearch users. If you understand the cost of keeping the context open for a long time and would like to pass a `scroll` value longer than an hour, set `ignore_scroll_restriction` to `TRUE`.

`intermediates_dir`
: When scrolling over search results, this function writes intermediate results to disk. By default, ‘es_search‘ will create a temporary directory in whatever working directory the function is called from. If you want to change this behavior, provide a path here. ‘es_search‘ will create and write to a temporary directory under whatever path you provide.

References

Elasticsearch 6 scrolling strategy

Examples

```
## Not run:

###=== Example 1: Get low-scoring food survey results ===###

query_body <- '{"query":{"filtered":{"filter":{"bool":{"must":[
               {"exists":{"field":"customer_comments"}},
               {"terms":{"overall_satisfaction":["very low","low"]}}]}}},
               "query":{"match_phrase":{"customer_comments":"food"}}}}'

# Execute the query, parse into a data.table
commentDT <- es_search(es_host = 'http://mydb.mycompany.com:9200'
                       , es_index = "survey_results"
                       , query_body = query_body
                       , scroll = "1m"
                       , n_cores = 4)

###=== Example 2: Time series agg features ===###

# Create query that will give you daily summary stats for revenue
query_body <- '{"query":{"filtered":{"filter":{"bool":{"must":[
               {"exists":{"field":"pmt_amount"}}]}}}},
          "aggs":{"timestamp":{"date_histogram":{"field":"timestamp","interval":"day"},
              "aggs":{"revenue":{"extended_stats":{"field":"pmt_amount"}}}}},"size":0}'

# Execute the query and get the result
resultDT <- es_search(es_host = "http://es.custdb.mycompany.com:9200"
```

```
                      , es_index = 'ticket_sales'
                      , query_body = query_body)

## End(Not run)
```

`get_fields` *Get the names and data types of the indexed fields in an index*

Description

For a given Elasticsearch index, return the mapping from field name to data type for all indexed fields.

Usage

```
get_fields(es_host, es_indices = "_all")
```

Arguments

`es_host`	A string identifying an Elasticsearch host. This should be of the form `[transfer_protocol][hostname]` For example, `'http://myindex.thing.com:9200'`.
`es_indices`	A character vector that contains the names of indices for which to get mappings. Default is `'_all'`, which means get the mapping for all indices. Names of indices can be treated as regular expressions.

Value

A data.table containing four columns: index, type, field, and data_type

Examples

```
## Not run:
# get the mapping for all indexed fields in the ticket_sales and customers indices
mappingDT <- get_fields(es_host = "http://es.custdb.mycompany.com:9200"
                              , es_indices = c("ticket_sales", "customers"))

## End(Not run)
```

parse_date_time *Parse date-times from Elasticsearch records*

Description

Given a data.table with date-time strings, this function converts those dates-times to type POSIXct with the appropriate time zone. Assumption is that dates are of the form "2016-07-25T22:15:19Z" where T is just a separator and the last letter is a military timezone.

This is a side-effect-free function: it returns a new data.table and the input data.table is unmodified.

Usage

```
parse_date_time(input_df, date_cols, assume_tz = "UTC")
```

Arguments

input_df	a data.table with one or more date-time columns you want to convert
date_cols	Character vector of column names to convert. Columns should have string dates of the form "2016-07-25T22:15:19Z".
assume_tz	Timezone to convert to if parsing fails. Default is UTC

References

https://www.timeanddate.com/time/zones/military

https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

Examples

```
# Sample es_search(), chomp_hits(), or chomp_aggs() output:
someDT <- data.table::data.table(id = 1:5
                          , company = c("Apple", "Apple", "Banana", "Banana", "Cucumber")
                          , timestamp = c("2015-03-14T09:26:53B", "2015-03-14T09:26:54B"
                                       , "2031-06-28T08:53:07Z", "2031-06-28T08:53:08Z"
                                                , "2000-01-01"))

# Note that the date field is character right now
str(someDT)

# Let's fix that!
someDT <- parse_date_time(input_df = someDT
                          , date_cols = "timestamp"
                          , assume_tz = "UTC")
str(someDT)
```

unpack_nested_data *Unpack a nested data.table*

Description

After calling a chomp_* function or es_search, if you had a nested array in the JSON, its corresponding column in the resulting data.table is a data.frame itself (or a list of vectors). This function expands that nested column out, adding its data to the original data.table, and duplicating metadata down the rows as necessary.

This is a side-effect-free function: it returns a new data.table and the input data.table is unmodified.

Usage

```
unpack_nested_data(chomped_df, col_to_unpack)
```

Arguments

chomped_df	a data.table
col_to_unpack	a character vector of length one: the column name to unpack

Examples

```
# A sample raw result from a hits query:
result <- '[{"_source":{"timestamp":"2017-01-01","cust_name":"Austin","details":{
"cust_class":"big_spender","location":"chicago","pastPurchases":[{"film":"The Notebook",
"pmt_amount":6.25},{"film":"The Town","pmt_amount":8.00},{"film":"Zootopia","pmt_amount":7.50,
"matinee":true}]}}},{"_source":{"timestamp":"2017-02-02","cust_name":"James","details":{
"cust_class":"peasant","location":"chicago","pastPurchases":[{"film":"Minions",
"pmt_amount":6.25,"matinee":true},{"film":"Rogue One","pmt_amount":10.25},{"film":"Bridesmaids",
"pmt_amount":8.75},{"film":"Bridesmaids","pmt_amount":6.25,"matinee":true}]}}},{"_source":{
"timestamp":"2017-03-03","cust_name":"Nick","details":{"cust_class":"critic","location":"cannes",
"pastPurchases":[{"film":"Aala Kaf Ifrit","pmt_amount":0,"matinee":true},{
"film":"Dopo la guerra (Apres la Guerre)","pmt_amount":0,"matinee":true},{
"film":"Avengers: Infinity War","pmt_amount":12.75}]}}}]'

# Chomp into a data.table
sampleChompedDT <- chomp_hits(hits_json = result, keep_nested_data_cols = TRUE)
print(sampleChompedDT)

# (Note: use es_search() to get here in one step)

# Unpack by details.pastPurchases
unpackedDT <- unpack_nested_data(chomped_df = sampleChompedDT
                                 , col_to_unpack = "details.pastPurchases")
print(unpackedDT)
```

Index

chomp_aggs, 2
chomp_hits, 3

es_search, 4

fromJSON, *3*

get_fields, 6

parse_date_time, 7

unpack_nested_data, 8