HA2: collection indexing using Elasticsearch (part 1)

- Download elasticsearch https://www.elastic.co/downloads/elasticsearch
- Install Python Elasticsearch client https://elasticsearch-py.readthedocs.io/
- Read documentation https://www.elastic.co/guide/en/elasticsearch/reference/8.6/

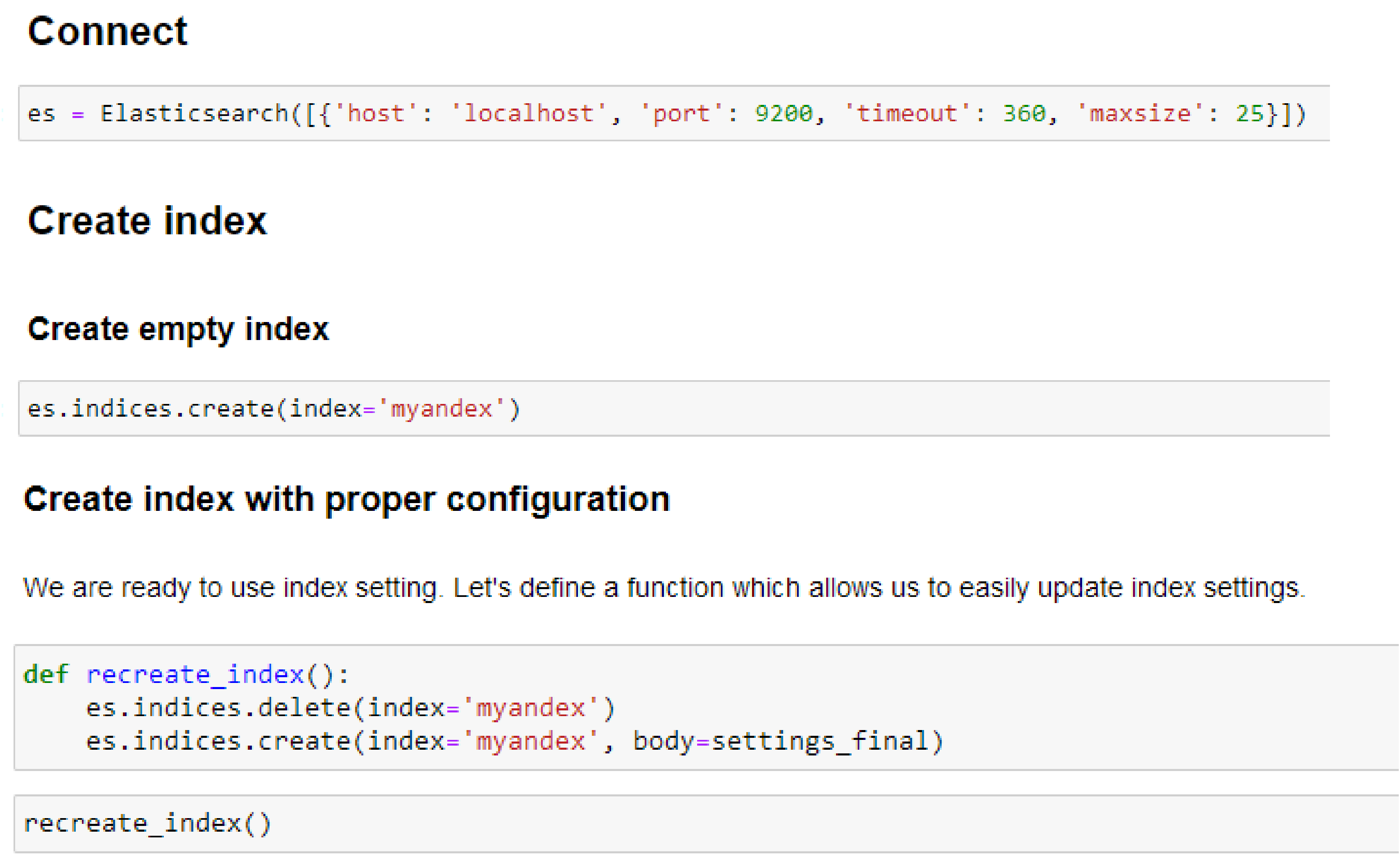
Steps to build a search engine

- Define configuration
- Create empty index
- Index collection
- Perform search
- (Evaluate results)

Use a tutorial example by Vladislav Korablinov as reference
*I didn’t check if it works with the latest Elasticsearch version

Main concepts

- Mappings: document structure
  very simple in our case: a single text field
- Analyzers: tokenization, filtering, normalization
  our case: whitespace tokenization, --/+ stemming
- Query-document matching
  BM25 (default)



Connect

```
es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'timeout': 360, 'maxsize': 25}])
```

Create index

Create empty index

```
es.indices.create(index='myandex')
```

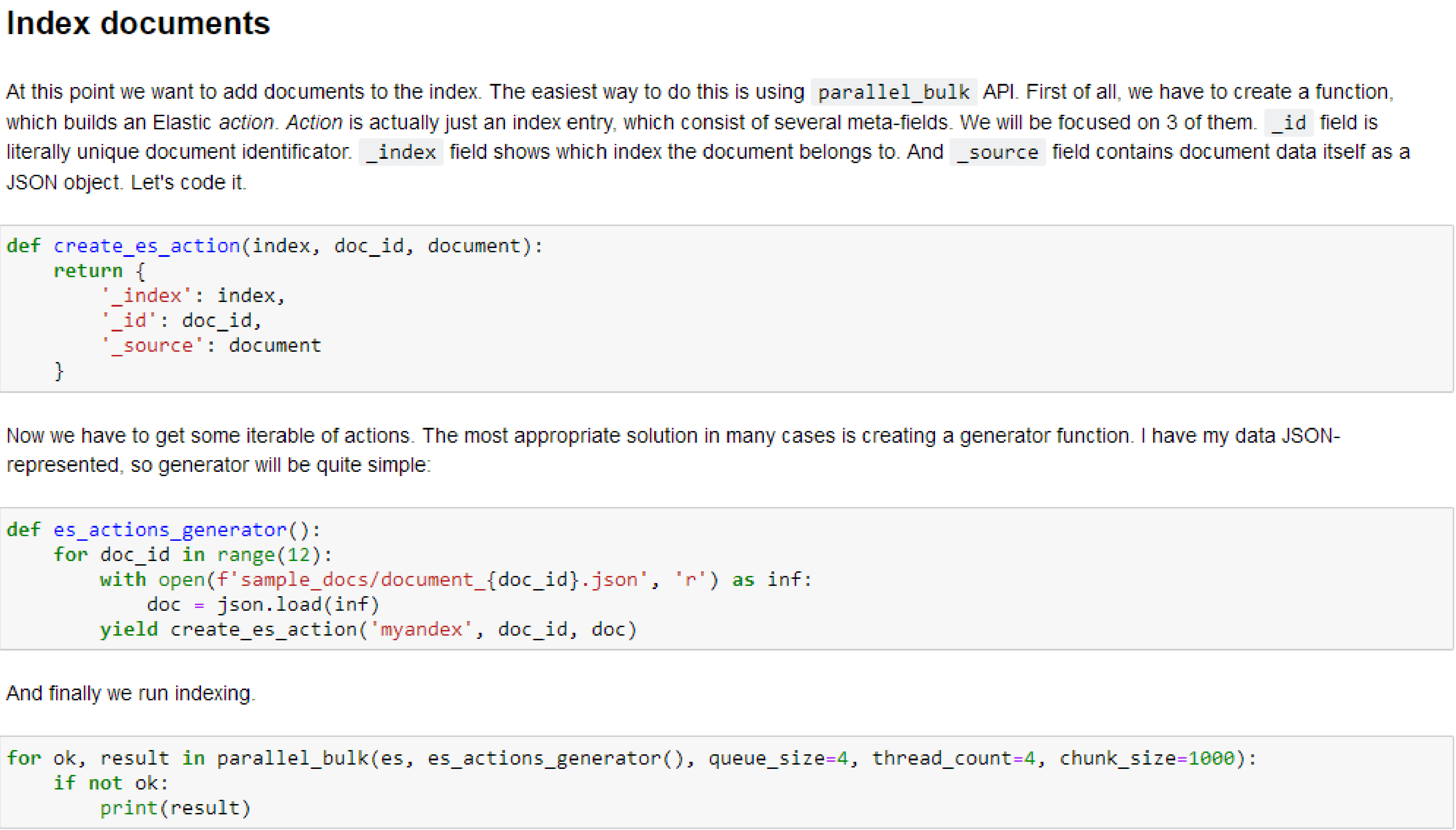
Create index with proper configuration

We are ready to use index setting. Let's define a function which allows us to easily update index settings.

```
def recreate_index():
    es.indices.delete(index='myandex')
    es.indices.create(index='myandex', body=settings_final)
```

```
recreate_index()
```

See configuration examples in Vladislav’s notebook



Index documents

At this point we want to add documents to the index. The easiest way to do this is using `parallel_bulk` API. First of all, we have to create a function, which builds an Elastic *action*. *Action* is actually just an index entry, which consist of several meta-fields. We will be focused on 3 of them. `_id` field is literally unique document identificator. `_index` field shows which index the document belongs to. And `_source` field contains document data itself as a JSON object. Let's code it.

```
def create_es_action(index, doc_id, document):
    return {
        '_index': index,
        '_id': doc_id,
        '_source': document
    }
```

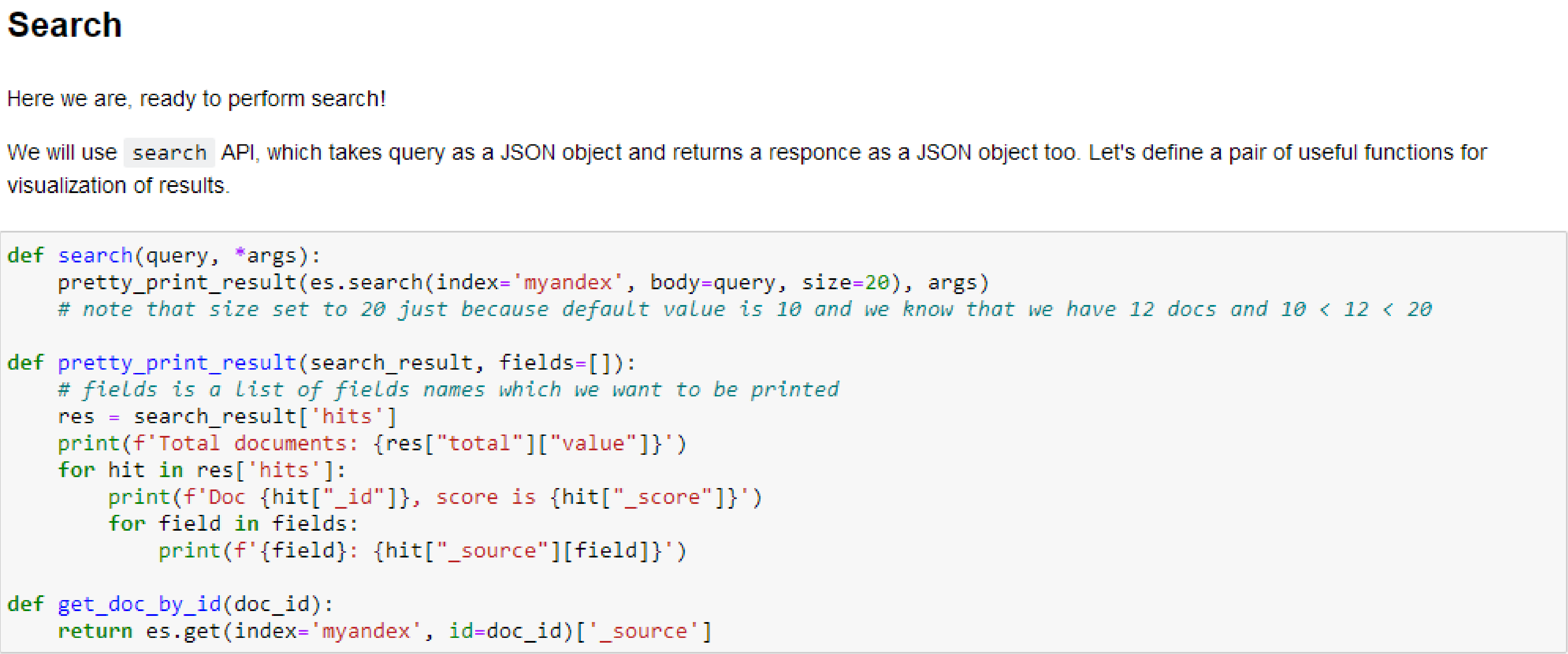
Now we have to get some iterable of actions. The most appropriate solution in many cases is creating a generator function. I have my data JSON-represented, so generator will be quite simple:

```
def es_actions_generator():
    for doc_id in range(12):
        with open(f'sample_docs/document_{doc_id}.json', 'r') as inf:
            doc = json.load(inf)
        yield create_es_action('myandex', doc_id, doc)
```

And finally we run indexing.

```
for ok, result in parallel_bulk(es, es_actions_generator(), queue_size=4, thread_count=4, chunk_size=1000):
    if not ok:
        print(result)
```

You can use `elasticsearch.helpers.bulk`



Search

Here we are, ready to perform search!

We will use `search` API, which takes query as a JSON object and returns a responce as a JSON object too. Let's define a pair of useful functions for visualization of results.

```
def search(query, *args):
    pretty_print_result(es.search(index='myandex', body=query, size=20), args)
    # note that size set to 20 just because default value is 10 and we know that we have 12 docs and 10 < 12 < 20

def pretty_print_result(search_result, fields=[]):
    # fields is a list of fields names which we want to be printed
    res = search_result['hits']
    print(f'Total documents: {res["total"]["value"]}')
    for hit in res['hits']:
        print(f'Doc {hit["_id"]}, score is {hit["_score"]}')
        for field in fields:
            print(f'{field}: {hit["_source"][field]}')

def get_doc_by_id(doc_id):
    return es.get(index='myandex', id=doc_id)['_source']
```

See query variants in Vladislav’s notebook

Task (for now)

- Index WikiIR **en1k** collection, estimate document indexing time
  - Without stemming
  - With stemming
- Run *test* queries, get top20 results for each query, estimate query execution time
- Save triples
  <queryID, docID, score>
  for two variants
- *add one more variant:
  - Lemmatized collection (don’t forget to lemmatize queries)
  - Boost phrase matches

Alternatives

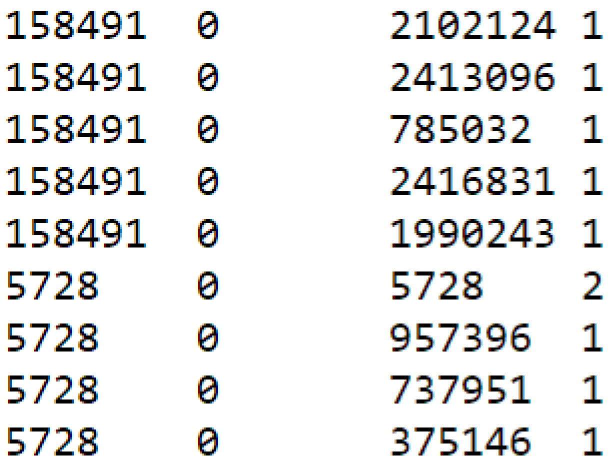
- gensim
  https://github.com/RaRe-Technologies/gensim/pull/3304
  not well documented, you have to take care of doc ids
- sklearn.feature_extraction.text.TfidfVectorizer
  tf.ifd only (not BM25)
- https://github.com/dorianbrown/rank_bm25
  presumably slow at query time
- https://github.com/AmenRa/retriv
  looks fine, but not matured yet, I guess

Part 2: Evaluation

- Format your runs in TREC format
- Use
  https://github.com/terrierteam/ir_measures (more formats, better documentation)
  or
  https://github.com/cvangysel/pytrec_eval (is in fact behind ir-measures)
- Calculate p@10, p@20, and MAP for your runs and the dataset creators' BM25 run (see test/BM25.res)

TREC formats

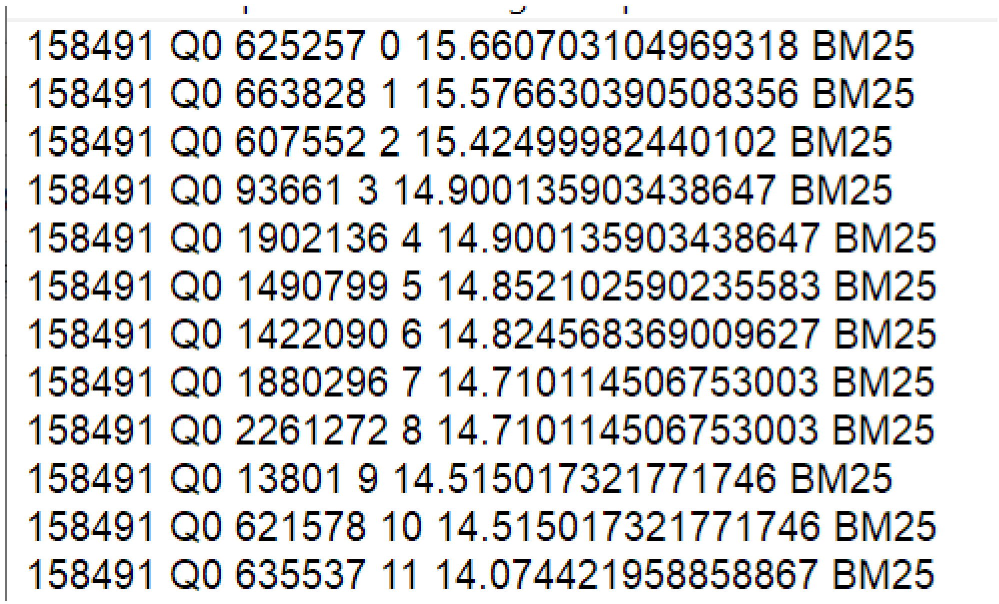
qrels



```
158491  0       2102124 1
158491  0       2413096 1
158491  0       785032  1
158491  0       2416831 1
158491  0       1990243 1
5728    0       5728    2
5728    0       957396  1
5728    0       737951  1
5728    0       375146  1
```

q_id not_used doc_id relevance_label

Runs



```
158491 Q0 625257 0 15.660703104969318 BM25
158491 Q0 663828 1 15.576630390508356 BM25
158491 Q0 607552 2 15.42499982440102 BM25
158491 Q0 93661 3 14.900135903438647 BM25
158491 Q0 1902136 4 14.900135903438647 BM25
158491 Q0 1490799 5 14.852102590235583 BM25
158491 Q0 1422090 6 14.824568369009627 BM25
158491 Q0 1880296 7 14.710114506753003 BM25
158491 Q0 2261272 8 14.710114506753003 BM25
158491 Q0 13801 9 14.515017321771746 BM25
158491 Q0 621578 10 14.515017321771746 BM25
158491 Q0 635537 11 14.074421958858867 BM25
```

q_id n_u doc_id rank score run_name