CE706 - Information Retrieval SU 2023

Assignment 1

Reg No: 2211433

Instructions for running your system:

The system is built with the help of Docker Desktop, a platform that allows to build, test and deploy applications quickly. Firstly, I loaded Elasticsearch version 5.6.16 in Docker's container and starting all the actions of the same. We then run our Elasticsearch on jupyter notebook by importing JSON and Elasticsearch libraries (elasticsearch5). After the connection is made, the localhost for Elasticsearch is launched on a web browser to check if it is up and running. We are using python programming language to build a code on further task of the assignment.

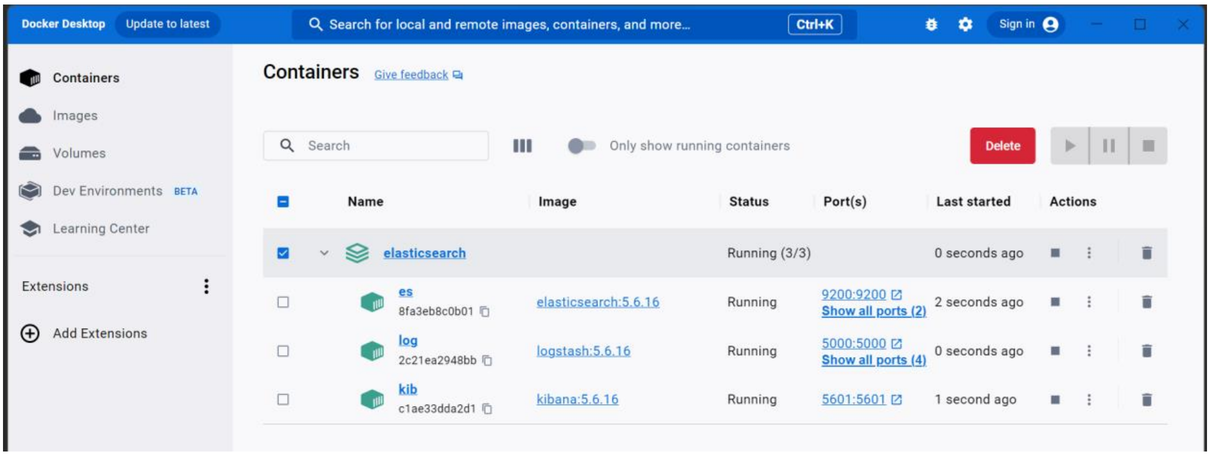


Fig 1.1 Docker Desktop Container with Elasticsearch

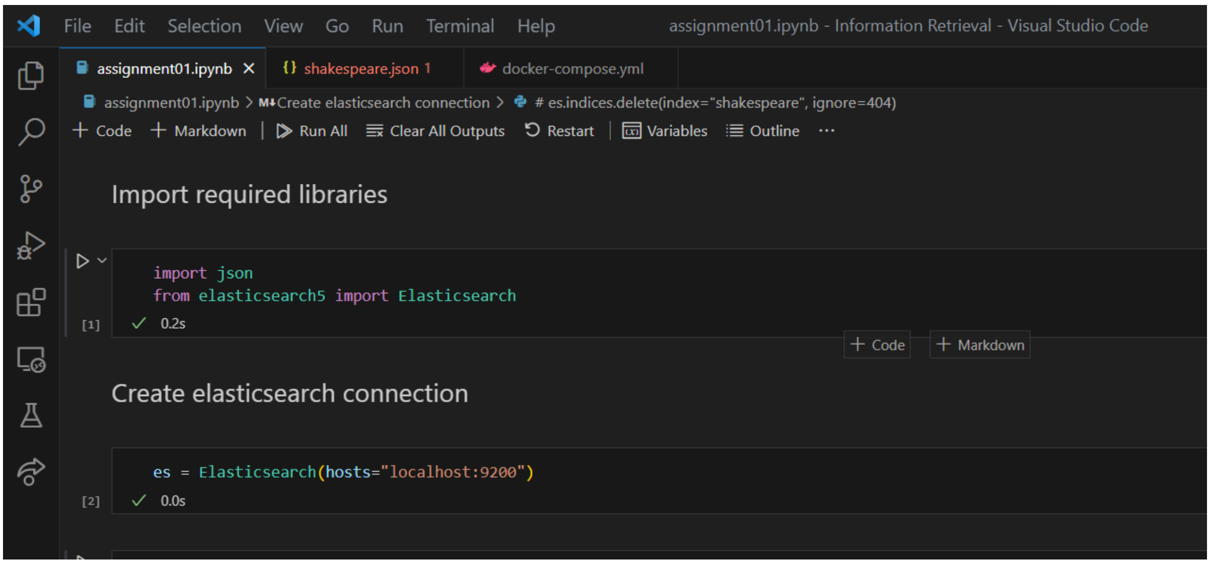


Fig 1.3 Necessary library import and connection with Elasticsearch localhost

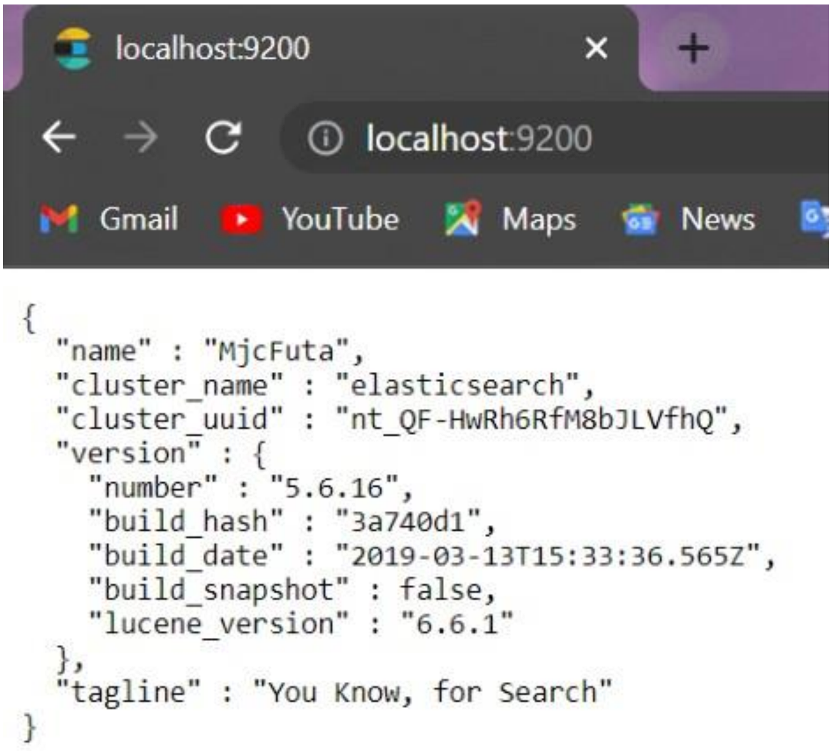


Fig 1.2 Elasticsearch server

Indexing:

The dataset is downloaded from Elasticsearch website with version 5.5 which is EOL date.

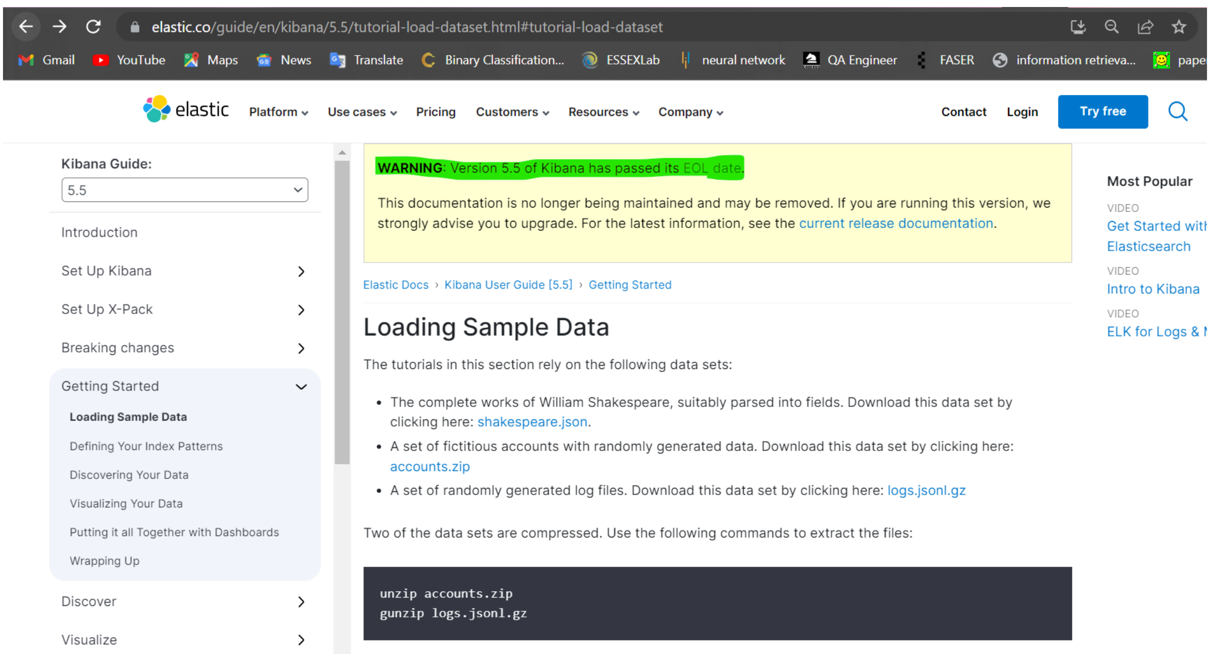


Fig 2.1 Dataset download

The Shakespeare dataset is not available under latest version of Elasticsearch to download.

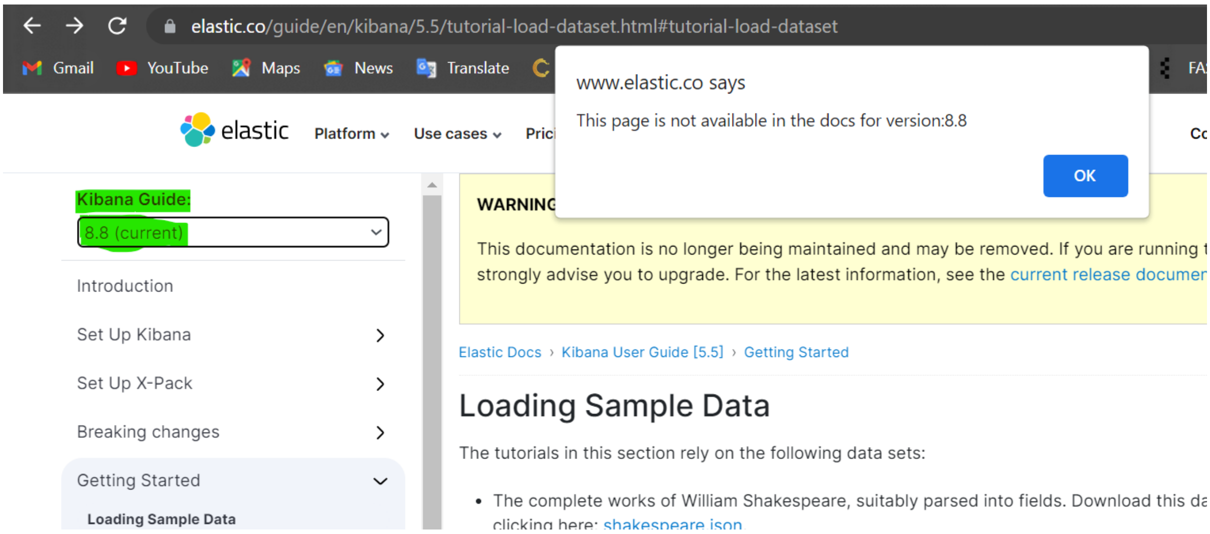
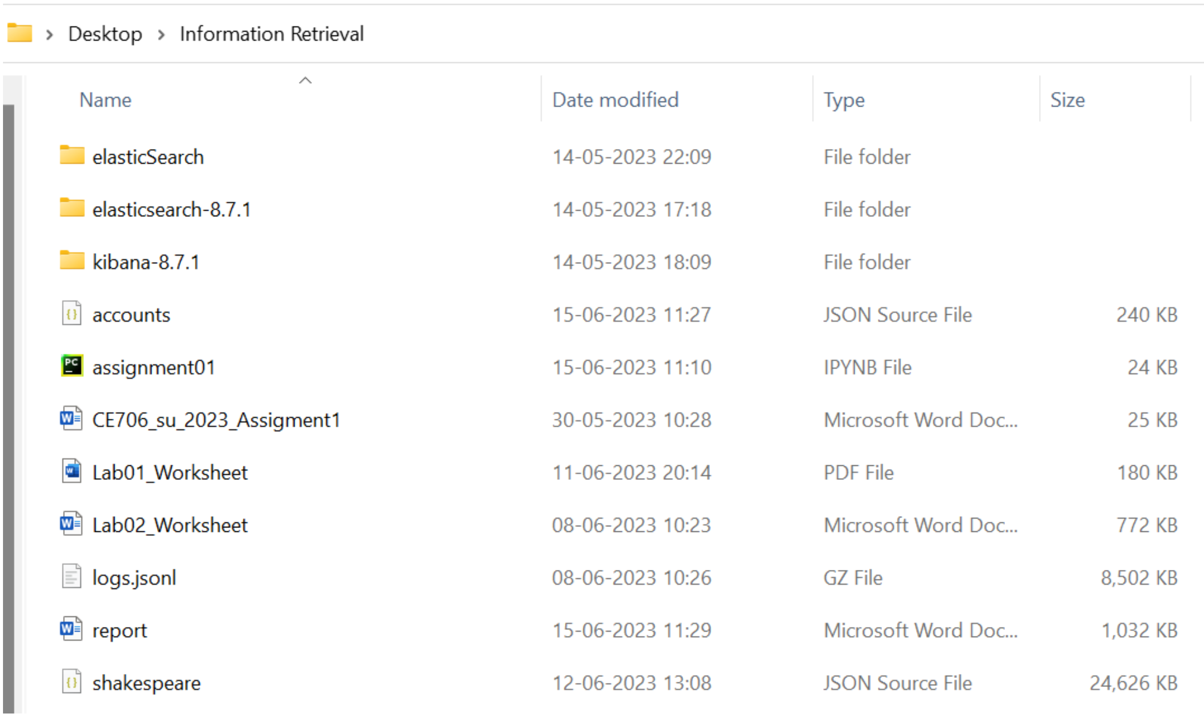


Fig 2.2 Dataset on latest version

The downloaded dataset file is manually extracted and saved in a separate folder created for my Information Retrieval Assignment 01.



Name	Date modified	Type	Size
elasticSearch	14-05-2023 22:09	File folder	
elasticsearch-8.7.1	14-05-2023 17:18	File folder	
kibana-8.7.1	14-05-2023 18:09	File folder	
accounts	15-06-2023 11:27	JSON Source File	240 KB
assignment01	15-06-2023 11:10	IPYNB File	24 KB
CE706_su_2023_Assigment1	30-05-2023 10:28	Microsoft Word Doc...	25 KB
Lab01_Worksheet	11-06-2023 20:14	PDF File	180 KB
Lab02_Worksheet	08-06-2023 10:23	Microsoft Word Doc...	772 KB
logs.jsonl	08-06-2023 10:26	GZ File	8,502 KB
report	15-06-2023 11:29	Microsoft Word Doc...	1,032 KB
shakespeare	12-06-2023 13:08	JSON Source File	24,626 KB

Fig 2.3 Necessary files and dataset for further analysis on Information Retrieval

Our dataset consists of three different fields for '_type': as 'act, 'line', 'scene' and multiple type mapping is unsupported in Elasticsearch from versions 6 and above. Hence, we are using the lower version 5.6.16. Before concluding on to the use of older v5.6.16, several attempts with other versions like v8.8 (current), v7.17, v6.6 were made but it did not result as expected.

Mapping the properties using the default mapping method of Elasticsearch for the dataset. This set of properties are stored in a variable, named "document_mapping". As mentioned earlier, for different fields of dataset, we are assigning all three fields to another variable and passing on the property's variable into this field's mapping.

Document mapping

```python
document_mappings = {
    "properties": {
        "line_id": {"type": "long"},
        "play_name": {"type": "text"},
        "line_number": {"type": "text"},
        "speaker": {"type": "text"},
        "speech_number": {"type": "text"},
        "speaker": {"type": "text"},
        "text_entry": {"type": "text"},
    }
}

mappings = {
    "mappings": {
        "line": document_mappings,
        "scene": document_mappings,
        "act": document_mappings,
    }
}

mappings
```

[3] ✓ 0.0s

Fig 2.3 Document Mapping for the dataset.

Creating Index with the existing functions of Elasticsearch indices and create by passing on the index value, body values that is stored under mapping section and Ignore set to 400 meaning if any error exists, do not fail the execution but continue running the program.

View the created index by making use of the .get_mapping() method for the created document mapping and the JSON.dumps() method that allows to convert a python object into an equivalent JSON object.

Adding Dataset document into index of Elasticsearch. The Shakespeare document is added into the index of created localhost connection with required parameters.

Add document to index

```
with open("shakespeare.json") as works_of_shakespeare:
    current_line = 1

    for line in works_of_shakespeare:
        if current_line % 2 > 0:
            index_info = json.loads(line.strip())
        else:
            document = json.loads(line.strip())
            index = index_info["index"]["_index"]
            doc_type = index_info["index"]["_type"]
            doc_id = index_info["index"]["_id"]

            es.index(index=index, doc_type=doc_type, id=doc_id, body=document)

        current_line += 1

        if current_line > 300:
            print("done")
            break
```

[ ]

```
index_stats = es.indices.stats(index=shakespeare_index)
index_stats["_all"]["primaries"]["docs"]["count"]
```

[ ]

Fig 2.4 Adding document to created index.

Tokenization and Normalization

In order to further analyse the data, we are making use of Custom analyser. For Tokenization, standard method is applied and for Normalization the Lowercase token filter is applied to convert all the text entries in the lowercase format. With the analyser, it first performs tokenization on the given number of documents i.e by converting each word of the text entry into tokens and filters it by changing all the mismatched cases of texts into alphabets of lowercase.

After custom analyser is created, a new copy of mapping for the custom analyser is created and the custom analyser is tagged along into this new mapping of document. Settings methods is used to configure the customer analyser into the mapping for all "_type" fields of the document.

Deleting the older index and create a new index with the above configured custom analyser. A test sample for the same has been mentioned in another section.

Test analyzer

```python
analyzer_output = es.indices.analyze(
    index=shakespeare_index,
    body={
        "text": "And BreatHe shORT-winded ACCENTS 112@12 of new broils",
        "analyzer": "standard",
    },
)
tokens = analyzer_output["tokens"]
for token in tokens:
    print(token["token"])
```

[20]

```
and
breathe
short
winded
accents
112
12
of
new
broils
```

Fig 3.1 Example of Tokenization and Normalization (case folding)

For the given above example of standard analyser, the text's output is by converting all the uppercase letters to lowercase and removes all the special characters.

Stemming or Morphological Analysis

For stemming analysis, I'm applying "porter_stem" method that removes common morphological and inflexional endings from the words of English. This method is added into the same mapping of tokenization and case folding but as a next step. An update of the mapping is configured using settings method for all

the fields of "_type" similar to the setting configuration of tokenization and stemming section.

Test analyzer

```python
analyzer_output = es.indices.analyze(
    index=shakespeare_index,
    body={
        "text": "There are many buses available for route NH77 playing running bathing eating quickly",
        "analyzer": "my_custom_analyzer",
    },
)
tokens = analyzer_output["tokens"]
for token in tokens:
    print(token["token"])
```

[32]

```
there
are
mani
buse
avail
for
rout
nh77
play
run
bath
eat
quick
```

Fig 4.1 Stemming/Morphological example

A minor mistake while working with custom analyser "Settings" was made which did not output the stemming on given text entry. i.e., Instead of the keyword "Settings" into the configuration I entered "Setting" which resulted as below.

Stemming or Morphological Analysis

```python
# create custom analyzer with tokenization and casefolding and stemming
custom_analyzer = {
    "analysis": {
        "analyzer": {
            "my_custom_analyzer": {
                "type": "custom",
                "tokenizer": "standard",
                "filter": ["lowercase", "snowball"],
            }
        }
    }
}

# update elastic search config

elastic_search_config["setting"] = custom_analyzer

elastic_search_config
```

[41]

Fig 4.2 Error on configure setting for stemming

Test analyzer

```
analyzer_output = es.indices.analyze(
    index=shakespeare_index,
    body={
        "text": "There are many buses available for route NH77 playing running bathing eating quickly",
        "analyzer": "my_custom_analyzer",
    },
)
tokens = analyzer_output["tokens"]
for token in tokens:
    print(token["token"])
```

[43]

```
there
are
many
buses
available
for
route
nh77
playing
running
bathing
eating
quickly
```

Fig 4.2 Output of error on configure setting for stemming

Selecting keywords

The selected keywords for our further analysis on dataset are “ngram”, “stopword removal” and finding “tf.idf” scores.

In existing custom analyser, I added “stop” keyword into the filter to remove the **stopwords** and the same is configured on the mapping of the document with Settings and a test on the mapped analyser is performed.

Removing stopwords

```python
# create custom analyzer with tokenization and casefolding and stemming
custom_analyzer = {
    "analysis": {
        "analyzer": {
            "my_custom_analyzer": {
                "type": "custom",
                "tokenizer": "standard",
                "filter": ["lowercase", "stop", "porter_stem"],
            }
        }
    }
}

# update elastic search config

elastic_search_config["settings"] = custom_analyzer

elastic_search_config
```

[27]

Fig 5.1 Selecting keywords – Stopword Removal

Test Analyzer for stopword removal

```python
analyzer_output = es.indices.analyze(
    index=shakespeare_index,
    body={"text": "The baby is playing 123!!!!", "analyzer": "my_custom_analyzer"},
)
tokens = analyzer_output["tokens"]
t = [token["token"] for token in tokens]
print(t)
```

[29]

```
... ['babi', 'plai', '123']
```

Fig 5.2 Example for stopword removal along with other filters

In the same custom analyser, I'm adding the **N-Gram** filter as "bigram" with minimum and maximum values set to 2 respectively assigned to "type" as "ngram". The updated custom analyser is then configured to mapping of the document using Settings. An example is mentioned below in Fig 5.4.

N-Gram

```
# # add stemming to custom analyzer

# create custom analyzer for n-gram

custom_analyzer = {
    "analysis": {
        "analyzer": {
            "my_custom_analyzer": {
                "type": "custom",
                "tokenizer": "standard",
                "filter":  ["lowercase", "stop", "porter_stem", "bigram"],
            }
        },
        "filter": {"bigram": {"type": "ngram", "min_gram": 2, "max_gram": 2}},
    }
}


# update elastic search config

elastic_search_config["settings"] = custom_analyzer

elastic_search_config
```

[2] ✓ 0.1s

Fig 5.3 Custom analyser update for N-Grams

Test analyzer for ngrams

```
analyzer_output = es.indices.analyze(
    index=shakespeare_index,
    body={"text": "Children are playing 123!!!!", "analyzer": "my_custom_analyzer"},
)
tokens = analyzer_output["tokens"]
t = [token["token"] for token in tokens]
print(t)
```

[35]

```
['ch', 'hi', 'il', 'ld', 'dr', 're', 'en', 'pl', 'la', 'ai', '12', '23']
```

Fig 5.4 Example for N-Gram tokenizer

In the above example, the text line is first performs standard tokenization, converts them to lowercase, removes all the special characters present in the line, performs stemming, removes English stopwords, and outputs a list of words in list of bigrams.

TD.IDF

After all the custom analysers are completed and executed successfully, we are then trying to configure tf.idf score using Elasticsearch similarity module. The term frequency of how many times a given word appears in the document and inverse documentary frequency calculation is the score that how rare your word is in the corpus. This score is obtained by the in-built functionality of Elasticsearch called Classic Similarity. I have stored this similarity methods as a dictionary to a variable and then passing on this dictionary variable to the index of mapping with different fields of text_entry.

Adding TF.IDF to the updated index with ngrams

```python
# add tfidf to mapping to text entry mapping in elasticsearch config

similarity_property = {"similarity": "classic"}

elastic_search_config["mappings"]["line"]["properties"]["text_entry"].update(
    similarity_property
)
```

[21] ✓ 0.0s

Fig 5.5 Similarity module TF.IDF

Note: For every time the custom analyser is updated with new token or filter i.e, updating custom analyser by adding porter_stem, stop, lowercase, I'm deleting the created indices and creating a new index each time as Update option to the index is not available.

Once all the task of pre-processing steps has been completed, I'm importing the Shakespeare dataset file into the index with 1000 lines in order to perform search queries. This step is basically Indexing 1000 documents from the Shakespeare dataset.

Searching

The Elasticsearch search queries used are search on full text, search with exact phrase, match exact phrase using AND operator on particular fields, search query to match part of phrase (prefix), sorting the field "line_id" in descending order and additionally Pagination and Filter part of phrase is also executed.

Search query for Full text

```python
query = {"query": {"match": {"text_entry": {"query": "pagans"}}}}

resp = es.search(index="shakespeare", body=query)
for hit in resp["hits"]["hits"]:
    print(hit)
```

[41] Python


```
{'_index': 'shakespeare', '_type': 'line', '_id': '26', '_score': 5.6923537, '_source': {'line_id': 27, 'play_name': 'Henry IV', 'speech_number': 1, 'line_number': '1.1.24', 'speaker': 'KING HENRY IV', 'text_entry': 'To chase
{'_index': 'shakespeare', '_type': 'line', '_id': '168', '_score': 5.594323, '_source': {'line_id': 169, 'play_name': 'Henry IV', 'speech_number': 17, 'line_number': '1.2.55', 'speaker': 'FALSTAFF', 'text_entry': 'wag, shall
{'_index': 'shakespeare', '_type': 'line', '_id': '42', '_score': 5.274379, '_source': {'line_id': 43, 'play_name': 'Henry IV', 'speech_number': 2, 'line_number': '1.1.40', 'speaker': 'WESTMORELAND', 'text_entry': 'Against th
{'_index': 'shakespeare', '_type': 'line', '_id': '427', '_score': 5.274379, '_source': {'line_id': 428, 'play_name': 'Henry IV', 'speech_number': 9, 'line_number': '1.3.100', 'speaker': 'HOTSPUR', 'text_entry': 'In single op
{'_index': 'shakespeare', '_type': 'line', '_id': '411', '_score': 4.9826007, '_source': {'line_id': 412, 'play_name': 'Henry IV', 'speech_number': 8, 'line_number': '1.3.84', 'speaker': 'KING HENRY IV', 'text_entry': 'Again
{'_index': 'shakespeare', '_type': 'line', '_id': '367', '_score': 4.844826, '_source': {'line_id': 368, 'play_name': 'Henry IV', 'speech_number': 6, 'line_number': '1.3.40', 'speaker': 'HOTSPUR', 'text_entry': 'He gave his
{'_index': 'shakespeare', '_type': 'line', '_id': '109', '_score': 4.64523, '_source': {'line_id': 110, 'play_name': 'Henry IV', 'speech_number': 9, 'line_number': '1.1.107', 'speaker': 'KING HENRY IV', 'text_entry': 'Than o
{'_index': 'shakespeare', '_type': 'line', '_id': '101', '_score': 4.6150813, '_source': {'line_id': 102, 'play_name': 'Henry IV', 'speech_number': 8, 'line_number': '1.1.99', 'speaker': 'WESTMORELAND', 'text_entry': 'The cr
{'_index': 'shakespeare', '_type': 'line', '_id': '29', '_score': 4.6001163, '_source': {'line_id': 30, 'play_name': 'Henry IV', 'speech_number': 1, 'line_number': '1.1.27', 'speaker': 'KING HENRY IV', 'text_entry': 'For our
{'_index': 'shakespeare', '_type': 'line', '_id': '499', '_score': 4.6001163, '_source': {'line_id': 500, 'play_name': 'Henry IV', 'speech_number': 22, 'line_number': '1.3.170', 'speaker': 'HOTSPUR', 'text_entry': 'The cords,
```


Fig 6.1 Search on full text for query "pagans" is resulted

Search query to match exact phrases

```python
match_phrase = {"query": {"match_phrase": {"text_entry": {"query": "thy love"}}}}

resp = es.search(index="shakespeare", body=match_phrase)
for hit in resp["hits"]["hits"]:
    print(hit)
```

[42] Python


```
{'_index': 'shakespeare', '_type': 'line', '_id': '333', '_score': 130.50565, '_source': {'line_id': 334, 'play_name': 'Henry IV', 'speech_number': 1, 'line_number': '1.3.8', 'speaker': 'KING HENRY IV', 'text_entry': 'And th
{'_index': 'shakespeare', '_type': 'line', '_id': '472', '_score': 123.457504, '_source': {'line_id': 473, 'play_name': 'Henry IV', 'speech_number': 16, 'line_number': '1.3.143', 'speaker': 'HOTSPUR', 'text_entry': 'He will,
{'_index': 'shakespeare', '_type': 'line', '_id': '295', '_score': 73.40764, '_source': {'line_id': 296, 'play_name': 'Henry IV', 'speech_number': 60, 'line_number': '1.2.180', 'speaker': 'PRINCE HENRY', 'text_entry': 'Well,
```


Fig 6.2 Search query to match exact phrase "thy love" on text_entry field is executed

Search query to match phrases on multipe fields using operator

```python
match_phrase = {
    "query": {
        "multi_match": {
            "query": "ever valiant",
            "operator": "and",
            "fields": ["speaker", "text_entry"],
        }
    }
}

resp = es.search(index="shakespeare", body=match_phrase)
for hit in resp["hits"]["hits"]:
    print(hit)
```

[43] Python


```
{'_index': 'shakespeare', '_type': 'line', '_id': '56', '_score': 10.594317, '_source': {'line_id': 57, 'play_name': 'Henry IV', 'speech_number': 4, 'line_number': '1.1.54', 'speaker': 'WESTMORELAND', 'text_entry': 'That ever
{'_index': 'shakespeare', '_type': 'line', '_id': '486', '_score': 8.840794, '_source': {'line_id': 487, 'play_name': 'Henry IV', 'speech_number': 19, 'line_number': '1.3.157', 'speaker': 'EARL OF WORCESTER', 'text_entry': 'L
{'_index': 'shakespeare', '_type': 'line', '_id': '344', '_score': 8.548342, '_source': {'line_id': 345, 'play_name': 'Henry IV', 'speech_number': 4, 'line_number': '1.3.19', 'speaker': 'KING HENRY IV', 'text_entry': 'The moo
{'_index': 'shakespeare', '_type': 'line', '_id': '29', '_score': 7.969216, '_source': {'line_id': 30, 'play_name': 'Henry IV', 'speech_number': 1, 'line_number': '1.1.27', 'speaker': 'KING HENRY IV', 'text_entry': 'For our a
{'_index': 'shakespeare', '_type': 'line', '_id': '499', '_score': 7.902654, '_source': {'line_id': 500, 'play_name': 'Henry IV', 'speech_number': 22, 'line_number': '1.3.170', 'speaker': 'HOTSPUR', 'text_entry': 'The cords,
{'_index': 'shakespeare', '_type': 'line', '_id': '84', '_score': 7.5642724, '_source': {'line_id': 85, 'play_name': 'Henry IV', 'speech_number': 7, 'line_number': '1.1.82', 'speaker': 'KING HENRY IV', 'text_entry': 'Amongst
{'_index': 'shakespeare', '_type': 'line', '_id': '233', '_score': 7.5218983, '_source': {'line_id': 234, 'play_name': 'Henry IV', 'speech_number': 39, 'line_number': '1.2.119', 'speaker': 'POINS', 'text_entry': 'to Canterbur
{'_index': 'shakespeare', '_type': 'line', '_id': '109', '_score': 7.0088778, '_source': {'line_id': 110, 'play_name': 'Henry IV', 'speech_number': 9, 'line_number': '1.1.107', 'speaker': 'KING HENRY IV', 'text_entry': 'Than
{'_index': 'shakespeare', '_type': 'line', '_id': '102', '_score': 6.9823723, '_source': {'line_id': 103, 'play_name': 'Henry IV', 'speech_number': 9, 'line_number': '1.1.100', 'speaker': 'KING HENRY IV', 'text_entry': 'But I
{'_index': 'shakespeare', '_type': 'line', '_id': '113', '_score': 6.9495363, '_source': {'line_id': 114, 'play_name': 'Henry IV', 'speech_number': 10, 'line_number': '', 'speaker': 'WESTMORELAND', 'text_entry': 'Enter the PR
```


Fig 6.3 Search query to match phrases on multiple fields using AND operator

Search query to match part of phrase with multiple fields

```python
match_part_phrase = {
    "query": {
        "multi_match": {
            "query": "hol",
            "fields": ["speaker", "text_entry"],
            "type": "phrase_prefix",
        }
    }
}

resp = es.search(index="shakespeare", body=match_part_phrase)
for hit in resp["hits"]["hits"]:
    print(hit)
```

[44] Python


```
{'_index': 'shakespeare', '_type': 'line', '_id': '72', '_score': 5.040043, '_source': {'line_id': 73, 'play_name': 'Henry IV', 'speech_number': 5, 'line_number': '1.1.70', 'speaker': 'KING HENRY IV', 'text_entry': 'On Holme
{'_index': 'shakespeare', '_type': 'line', '_id': '185', '_score': 5.040043, '_source': {'line_id': 186, 'play_name': 'Henry IV', 'speech_number': 26, 'line_number': '1.2.72', 'speaker': 'PRINCE HENRY', 'text_entry': 'What s
{'_index': 'shakespeare', '_type': 'line', '_id': '67', '_score': 4.70306, '_source': {'line_id': 68, 'play_name': 'Henry IV', 'speech_number': 5, 'line_number': '1.1.65', 'speaker': 'KING HENRY IV', 'text_entry': 'Betwixt t
{'_index': 'shakespeare', '_type': 'line', '_id': '57', '_score': 4.2323227, '_source': {'line_id': 58, 'play_name': 'Henry IV', 'speech_number': 4, 'line_number': '1.1.55', 'speaker': 'WESTMORELAND', 'text_entry': 'At Holme
{'_index': 'shakespeare', '_type': 'line', '_id': '104', '_score': 4.0265493, '_source': {'line_id': 105, 'play_name': 'Henry IV', 'speech_number': 9, 'line_number': '1.1.102', 'speaker': 'KING HENRY IV', 'text_entry': 'Our
{'_index': 'shakespeare', '_type': 'line', '_id': '26', '_score': 3.8803897, '_source': {'line_id': 27, 'play_name': 'Henry IV', 'speech_number': 1, 'line_number': '1.1.24', 'speaker': 'KING HENRY IV', 'text_entry': 'To chas
{'_index': 'shakespeare', '_type': 'line', '_id': '54', '_score': 3.628461, '_source': {'line_id': 55, 'play_name': 'Henry IV', 'speech_number': 4, 'line_number': '1.1.52', 'speaker': 'WESTMORELAND', 'text_entry': 'On Holy-r
{'_index': 'shakespeare', '_type': 'line', '_id': '60', '_score': 3.6209426, '_source': {'line_id': 61, 'play_name': 'Henry IV', 'speech_number': 4, 'line_number': '1.1.58', 'speaker': 'WESTMORELAND', 'text_entry': 'And shap
{'_index': 'shakespeare', '_type': 'line', '_id': '374', '_score': 3.6209426, '_source': {'line_id': 375, 'play_name': 'Henry IV', 'speech_number': 6, 'line_number': '1.3.47', 'speaker': 'HOTSPUR', 'text_entry': 'With many h
{'_index': 'shakespeare', '_type': 'line', '_id': '74', '_score': 3.5738244, '_source': {'line_id': 75, 'play_name': 'Henry IV', 'speech_number': 5, 'line_number': '1.1.72', 'speaker': 'KING HENRY IV', 'text_entry': 'To beat
```


Fig 6.4 Search query to match part of phrase on multiple fields