Elasticsearch

- **Elasticsearch** is a distributed search and analytics engine designed to handle large-scale data.
- It can scale horizontally by adding more nodes to the cluster, making it capable of handling high-volume data and providing fast search responses.
- ES stores and indexes data in the form of JSON documents.
  - Each document contains one or more fields with their corresponding values.
  - Data is indexed in ES by specyfying an index, type and ID for each document.
- ES offers excellent search capabilities, including full-text search, filtering, aggregation etc.
- It provides a Query DSL (Domain-Specific Language) that allows you to construct complex queries using JSON-like syntax.
- It indexes and analyzes data to enable fast and accurate search results across various types of documents.

For more detailed information about Elasticsearch and its features, you can visit the official website: **Elasticsearch Official Website** (https://www.elastic.co/elasticsearch/)

Elasticsearch Python Library

- The **Python Elasticsearch** library is the official Python client for Elasticsearch.
- It provides a high-level and low-level interface to interact with Elasticsearch
- You can install elasticsearch using `pip3 install elasticsearch` from your terminal or `!pip3 install elasticsearch` in notebook :)

In [1]:

```
from getpass import getpass
from elasticsearch import Elasticsearch

config = {
    "ES_USER": "elastic",
    "ES_PASS": getpass("ES password: ")
}
```

```
ES password: ••••••••
```

In [2]:

```
%store -r data
data[0:5]
```

Out[2]:

```
[{'id': 1,
  'title': 'NumPy',
  'description': 'NumPy is a powerful python library with many funct
ions for creating and manipulating multi-dimensional arrays and matr
ices.'},
 {'id': 2,
  'title': 'Pandas',
  'description': 'Pandas is a Python library for data manipulation a
nd analysis. It provides data structures for efficient storage of da
ta and high-level manipulations.'},
 {'id': 3,
  'title': 'Scikit-Learn',
  'description': 'Scikit-Learn is a popular library for machine lear
ning in Python. It provides tools to build, train, evaluate, and dep
loy machine learning algorithms.'},
 {'id': 4,
  'title': 'Matplotlib',
  'description': 'Matplotlib is a Python plotting library for creati
ng publication quality plots. It can produce line graphs, histogram
s, power spectra, bar charts, and more.'},
 {'id': 5,
  'title': 'Seaborn',
  'description': 'Seaborn is a graphical library in Python for drawi
ng statistical graphics. It provides a high level interface for draw
ing attractive statistical graphics.'}]
```

Elasticsearch Client class represents the Elasticsearch client and is used to establish a connection with an Elasticsearch cluster. You can create an instance of the client by specifying the Elasticsearch host and port.

In [3]:

```
es = Elasticsearch(
    cloud_id=getpass("cloud id: "),
    basic_auth=(config['ES_USER'], config['ES_PASS']),
    request_timeout=60)
```

```
cloud id: ········
```

To check the connection with ES host, you can use `es.ping()` :

In [4]:

```
if es.ping():
    print("Connected to Elasticsearch.")
else:
    print("Failed to connect to Elasticsearch.")
```

```
Connected to Elasticsearch.
```

- Instead of using host and id, I am using cloud_id and basic_auth with username and password.
- To establish a connection to Elastic Cloud using the Python Elasticsearch client, it is recommended to utilize the cloud_id parameter. You can locate this value on the "Manage Deployment" page, which becomes accessible after creating a cluster. In Kibana, you can find it in the top-left corner of the page.

Index Management

- Elasticsearch library provides functions for managing indices such as creating, deleting, checking the existence of an index, and more.

In [5]:

```
# Create an index
es.indices.create(index='test')

# Check if an index exists
if es.indices.exists(index='test'):
    print("Index exists")
```

```
Index exists
```

In [6]:

```
# Delete an index
es.indices.delete(index='test')

if es.indices.exists(index='test'):
    print("Index exists")
else:
    print("Index doesn't exist")
```

```
Index doesn't exist
```

Document Indexing

- You can index documents in Elasticsearch using the index function.
- By indexing a document in ES you should understand - adding data to the database in a structured manner, like filling information in folders, so that it can be quickly searched and retrieved later on.
- This allows Elasticsearch to efficiently organize and find specific information, making it easier to work with large amounts of data.
- The document is represented as a Python dictionary and is associated with an index, type, and an optional ID.

In [7]:

```
# Create an example document
example_doc = {
    "title": 'Example Document',
    "content": "Content of an example document."
}

# Create an index
es.indices.create(index='example')

# Check if an index exists
if es.indices.exists(index='example'):
    print("Index exists")
else:
    print("Index doesn't exist.")

# Index a document
es.index(index='example', id=1, document=example_doc)
```

```
Index exists
```

Out[7]:

```
ObjectApiResponse({'_index': 'example', '_id': '1', '_version': 1,
'result': 'created', '_shards': {'total': 2, 'successful': 1, 'faile
d': 0}, '_seq_no': 0, '_primary_term': 1})
```

- `_index` : Indicates the name of the index where the document was indexed, in this case, 'example'.
- `_id` : Represents the unique identifier assigned to the indexed document, which is '1' in this case.
- `_version` : Denotes the version of the document after indexing, which is '1'.
- `result` : Indicates the result of the indexing operation. In this case, it is 'created', indicating that the document was successfully created and indexed.
- `_shards` : Provides information about the number of shards involved in the indexing process. Shards are smaller units of the index distributed across nodes in a cluster. In this case, 'total' is 2, indicating that the operation involved two shards, and 'successful' is 2, meaning the indexing was successful on all shards.
- `_seq_no` and `_primary_term` : These terms relate to Elasticsearch's internal versioning system, which helps maintain consistency and handle conflicts in distributed environments. They represent specific details about the internal versioning of the document.

Document Retrieval

- You can retrieve documents from ES using various methods like `get` , `search` etc.
  - These methods allow you to query and filter the documents based on specific criteria.

In [8]:

```
# Get a document by ID
result = es.get(index='example', id=1)
print(result)
```

```
{'_index': 'example', '_id': '1', '_version': 1, '_seq_no': 0, '_pri
mary_term': 1, 'found': True, '_source': {'title': 'Example Documen
t', 'content': 'Content of an example document.'}}
```

In [9]:

```
doc = result['_source']
print(doc)
```

```
{'title': 'Example Document', 'content': 'Content of an example docu
ment.'}
```

- `doc = result['_source']` is used to access the actual source data of the document from the response, allowing you to work directly with the document's fields and values without additional parsing or extraction steps.

In [10]:

```
# Search documents
query = {
        'match': {
            'title': 'Example Document'
        }
    }

results = es.search(index='example', query=query)
hits = results['hits']['hits']
```

In [11]:

```
print(results)
```

```
{'took': 2, 'timed_out': False, '_shards': {'total': 1, 'successfu
l': 1, 'skipped': 0, 'failed': 0}, 'hits': {'total': {'value': 1, 'r
elation': 'eq'}, 'max_score': 0.5753642, 'hits': [{'_index': 'exampl
e', '_id': '1', '_score': 0.5753642, '_source': {'title': 'Example D
ocument', 'content': 'Content of an example document.'}}]}}
```

In [12]:

```
print(hits)
```

```
[{'_index': 'example', '_id': '1', '_score': 0.5753642, '_source':
{'title': 'Example Document', 'content': 'Content of an example docu
ment.'}}]
```

In [13]:

```
es.indices.delete(index='example')

# Check if an index exists
if es.indices.exists(index='example'):
    print("Index exists")
else:
    print("Index doesn't exist.")
```

```
Index doesn't exist.
```

Bulk Operations

- The library provides the bulk function to perform bulk operations like indexing, updating, deleting multiple documents in a single API call, which can significantly improve indexing performance.
- The bulk API accepts a list of action items, where each item represents a specific operation to be performed on a document.
- Each action item consists of a combination of an operation (index, update, delete) and the corresponding document data.
- You can include multiple action items in a single bulk request to perform various operations simultaneously.

I will use `data` list of dictionaries as my documents to create bulk actions. We will need another index for this purpose.

In [14]:

```
data[0:5]
```

Out[14]:


```
[{'id': 1,
  'title': 'NumPy',
  'description': 'NumPy is a powerful python library with many funct
ions for creating and manipulating multi-dimensional arrays and matr
ices.'},
 {'id': 2,
  'title': 'Pandas',
  'description': 'Pandas is a Python library for data manipulation a
nd analysis. It provides data structures for efficient storage of da
ta and high-level manipulations.'},
 {'id': 3,
  'title': 'Scikit-Learn',
  'description': 'Scikit-Learn is a popular library for machine lear
ning in Python. It provides tools to build, train, evaluate, and dep
loy machine learning algorithms.'},
 {'id': 4,
  'title': 'Matplotlib',
  'description': 'Matplotlib is a Python plotting library for creati
ng publication quality plots. It can produce line graphs, histogram
s, power spectra, bar charts, and more.'},
 {'id': 5,
  'title': 'Seaborn',
  'description': 'Seaborn is a graphical library in Python for drawi
ng statistical graphics. It provides a high level interface for draw
ing attractive statistical graphics.'}]
```

In [15]:

```
from elasticsearch.helpers import bulk, BulkIndexError

# Create an index
py_indexname = 'py-libraries'
es.indices.create(index=py_indexname)

# Check if an index exists
if es.indices.exists(index=py_indexname):
    print("Index exists")
else:
    print("Index doesn't exist.")

actions = [
    {"_index": py_indexname,
     "_id": doc["id"],
     "_source": {
          "library": doc["title"],
          "description": doc["description"]}
    }
    for doc in data
]

try:
    bulk(es, actions)
    print("Data successfully indexed in the destination index.")
except BulkIndexError as e:
    print("Failed to index documents:")
    for err in e.errors:
        print(err)
```

```
Index exists
Data successfully indexed in the destination index.
```

The `bulk()` method takes the Elasticsearch client instance (es), the list of actions (actions), and the index name. It returns a response containing information about the bulk operation.

In [16]:

```
query = {
        "match_all": {}
}

es.count(index=py_indexname, query=query)
```

Out[16]:

```
ObjectApiResponse({'count': 53, '_shards': {'total': 1, 'successfu
l': 1, 'skipped': 0, 'failed': 0}})
```

In [17]:

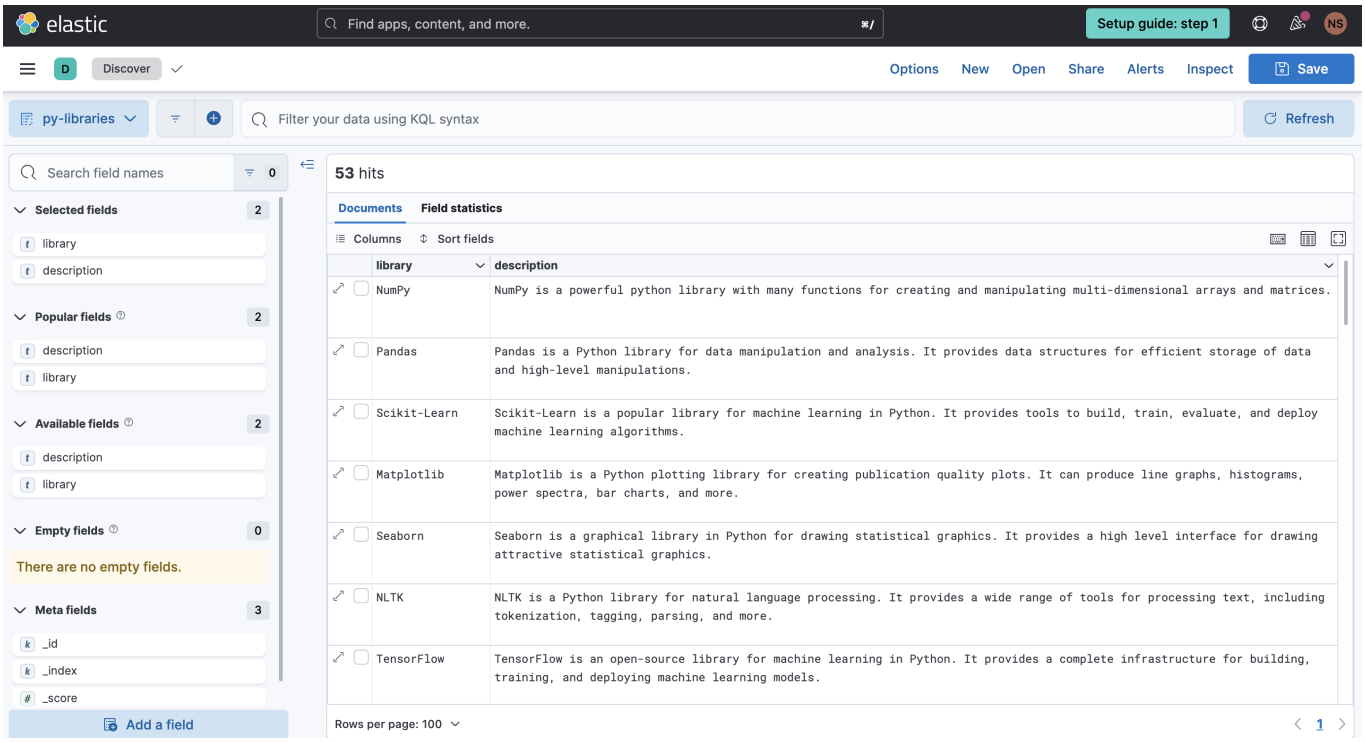
```
# Execute the search query
response = es.search(index=py_indexname, query=query, size=100)

# Extract the results
results = response["hits"]["hits"]

# Print the documents
for result in results:
    print(result["_source"])
```

{'library': 'NumPy', 'description': 'NumPy is a powerful python library with many functions for creating and manipulating multi-dimensional arrays and matrices.'}
{'library': 'Pandas', 'description': 'Pandas is a Python library for data manipulation and analysis. It provides data structures for efficient storage of data and high-level manipulations.'}
{'library': 'Scikit-Learn', 'description': 'Scikit-Learn is a popular library for machine learning in Python. It provides tools to build, train, evaluate, and deploy machine learning algorithms.'}
{'library': 'Matplotlib', 'description': 'Matplotlib is a Python plotting library for creating publication quality plots. It can produce line graphs, histograms, power spectra, bar charts, and more.'}
{'library': 'Seaborn', 'description': 'Seaborn is a graphical library in Python for drawing statistical graphics. It provides a high level interface for drawing attractive statistical graphics.'}
{'library': 'NLTK', 'description': 'NLTK is a Python library for natural language processing. It provides a wide range of tools for processing text, including tokenization, tagging, parsing, and more.'}
{'library': 'TensorFlow', 'description': 'TensorFlow is an open-source library for machine learning in Python. It provides a complete infrastructure for building, training, and deploying machine learning models.'}
{'library': 'Keras', 'description': 'Keras is an API for building neural networks in Python. It provides a simple and efficient way to create, train, and evaluate deep learning models.'}
{'library': 'sciPy', 'description': 'SciPy is a library for scientific computing in Python. It provides a range of tools for numerical integration, linear algebra, optimization, and more.'}
{'library': 'OpenCV', 'description': 'OpenCV is a library for computer vision in Python. It provides functions for image analysis, feature detection, and more.'}
{'library': 'PyTorch', 'description': 'PyTorch is an open-source machine learning library for Python. It provides a wide range of tools for developing, training, and evaluating neural networks.'}
{'library': 'BeautifulSoup', 'description': 'BeautifulSoup is a library for extracting data from HTML and XML documents. It provides a simple way to extract structured information from web pages.'}
{'library': 'Requests', 'description': 'Requests is a library for making HTTP requests in Python. It provides features to handle cookies, redirects, connection timeouts, and more.'}
{'library': 'Flask', 'description': 'Flask is a micro web framework for Python. It provides the tools to quickly build lightweight web applications.'}
{'library': 'pyspark', 'description': 'pyspark is a library for distributed computing in Python. It provides tools for parallel processing on clusters of machines.'}
{'library': 'Pillow', 'description': 'Pillow is a library for manipulating images in Python. It provides a range of features for manipulating, resizing, and transforming images.'}
{'library': 'Scrapy', 'description': 'Scrapy is a library for extracting data from web pages. It provides a convenient way to crawl websites and extract structured information.'}
{'library': 'pygame', 'description': 'pygame is a library for creating games in Python. It provides tools for making graphical user interfaces, playing sounds, and animating art.'}
{'library': 'django', 'description': 'Django is a web framework for Python. It provides the tools and libraries for building powerful web applications.'}
{'library': 'statsmodels', 'description': 'statsmodels is a library for statistical analysis in Python. It provides tools for fitting and testing statistical models.'}

{'library': 'pytz', 'description': 'pytz is a library for working wi
th time zones in Python. It provides tools to convert times between
different time zones.'}
{'library': 'xarray', 'description': 'xarray is a library for analyz
ing multi-dimensional arrays and datasets in Python. It provides a p
owerful framework for data analysis and visualization.'}
{'library': 'bokeh', 'description': 'Bokeh is a library for creating
interactive plots and dashboards in Python. It provides a high level
interface for drawing attractive graphics.'}
{'library': 'pyglet', 'description': 'pyglet is a library for multim
edia programming in Python. It provides features for playing 3D grap
hics and other multimedia.'}
{'library': 'Sphinx', 'description': 'Sphinx is a library for creati
ng documentation in Python. It provides tools for building, writing,
and maintaining comprehensive documentation.'}
{'library': 'pytest', 'description': 'pytest is a library for testin
g Python code. It provides tools for writing and running tests in an
automated way.'}
{'library': 'networkx', 'description': 'networkx is a Library for an
alyzing networks and graphs in Python. It provides tools for constru
cting, analyzing, and visualizing graphs.'}
{'library': 'NumExpr', 'description': 'NumExpr is a library for effi
ciently calculating numerical expressions in Python. It provides an
array-oriented approach for processing large datasets.'}
{'library': 'jupyter', 'description': 'Jupyter is a library for inte
ractive computing in Python. It provides tools for creating document
s containing live code, equations, and visualizations.'}
{'library': 'SymPy', 'description': 'SymPy is a library for symbolic
mathematics in Python. It provides tools for solving equations, mani
pulating expressions, and performing symbolic calculations.'}
{'library': 'scikit-image', 'description': 'scikit-image is a librar
y for image processing in Python. It provides algorithms for analyzi
ng, transforming, and manipulating images.'}
{'library': 'pySerial', 'description': 'pySerial is a library for co
mmunicating with serial ports in Python. It provides tools for sendi
ng and receiving data on serial and USB connections.'}
{'library': 'lxml', 'description': 'lxml is a library for processing
XML and HTML documents in Python. It provides powerful features for
parsing, validating, and manipulating XML and HTML documents.'}
{'library': 'Paramiko', 'description': 'Paramiko is a library for se
cure file transfers in Python. It provides an SSH2 protocol implemen
tation for connecting to remote machines.'}
{'library': 'PyYAML', 'description': 'PyYAML is a library for workin
g with YAML files in Python. It provides features for parsing, creat
ing, modifying, and saving YAML documents.'}
{'library': 'Numba', 'description': 'Numba is a library for optimizi
ng numerical code in Python. It provides tools for compiling Python
code into faster native instructions.'}
{'library': 'SciPy.optimize', 'description': 'SciPy.optimize is a li
brary for optimizing functions in Python. It provides a range of alg
orithms for finding minima and maxima of functions.'}
{'library': 'cProfile', 'description': 'cProfile is a library for pr
ofiling Python code. It provides a simple way to measure execution t
ime and identify bottlenecks in code.'}
{'library': 'SystemML', 'description': 'SystemML is a library for ma
chine learning in Python. It provides a high-level API for developin
g, training, and evaluating machine learning models.'}
{'library': 'PyODBC', 'description': 'PyODBC is a library for workin
g with databases in Python. It provides a high-level interface for c
onnecting to and querying relational databases.'}
{'library': 'SymEngine', 'description': 'SymEngine is a library for

symbolic computation in Python. It provides a unified interface for manipulating mathematical expressions and performing symbolic calculations.'}
{'library': 'ggplot', 'description': 'ggplot is a library for creating beautiful graphics in Python. It provides a high level interface for drawing statistical graphics with layered components.'}
{'library': 'Shapely', 'description': 'Shapely is a library for manipulating and analyzing geometric objects in Python. It provides features for calculating areas and distances, constructing shapes, and more.'}
{'library': 'pygsheets', 'description': 'pygsheets is a library for working with Google Sheets in Python. It provides high-level functions for interacting with Google Sheets spreadsheets.'}
{'library': 'SimpleCV', 'description': 'SimpleCV is a library for computer vision in Python. It provides functions for image processing, feature detection, and more.'}
{'library': 'pandasql', 'description': 'pandasql is a library for writing SQL queries in Python. It provides an elegant way to query pandas data frames using SQL syntax.'}
{'library': 'twisted', 'description': 'twisted is a library for asynchronous network programming in Python. It provides a wide range of tools for building and working with asynchronous services.'}
{'library': 'igraph', 'description': 'igraph is a library for analyzing graphs and networks in Python. It provides algorithms for measuring centrality, traversing graphs, and more.'}
{'library': 'PyMC3', 'description': 'PyMC3 is a library for probabilistic programming in Python. It provides high-level tools for building, fitting, and evaluating probabilistic models.'}
{'library': 'Biopython', 'description': 'Biopython is a library for working with biological data in Python. It provides tools for reading and writing sequence data, performing sequence analysis, and more.'}
{'library': 'cffi', 'description': 'cffi is a library for calling C functions from Python. It provides support for interacting with dynamic libraries and low-level operations.'}
{'library': 'PyQt', 'description': 'PyQt is a library for building graphical user interfaces in Python. It provides a comprehensive set of widgets and other GUI components.'}
{'library': 'pyspark.sql', 'description': 'pyspark.sql is a library for working with structured data in Python. It provides a high-level interface for processing data frames and running SQL queries.'}

We can also see our data in Elasticsearch Kibana "Discover" tab



Aggregations

- ES supports aggregations to perform analytics and gather insights from the data.
- Library provides functions to build and execute aggregations.

In [18]:

```
# Create an index
indexname = 'prices'
es.indices.create(index=indexname)

# Check if an index exists
if es.indices.exists(index=indexname):
    print("Index exists")
else:
    print("Index doesn't exist.")

actions = [
    {"_index": indexname,
     "_id": i,
     "_source": {
          "product": "example" + str(i),
          "price": 3.20 + i*0.75}
    }
    for i in range(20)
]

try:
    bulk(es, actions)
    print("Data successfully indexed in the destination index.")
except BulkIndexError as e:
    print("Failed to index documents:")
    for err in e.errors:
        print(err)
```

```
Index exists
Data successfully indexed in the destination index.
```

In [19]:

```
query = {
        "match_all": {}
}

es.count(index=indexname, query=query)
```

Out[19]:

```
ObjectApiResponse({'count': 20, '_shards': {'total': 1, 'successfu
l': 1, 'skipped': 0, 'failed': 0}})
```

In [20]:

```
# Execute the search query
response = es.search(index=indexname, query=query, size=20)

# Extract the results
results = response["hits"]["hits"]

# Print the documents
for result in results:
    print(result["_source"])
```

```
{'product': 'example0', 'price': 3.2}
{'product': 'example1', 'price': 3.95}
{'product': 'example2', 'price': 4.7}
{'product': 'example3', 'price': 5.45}
{'product': 'example4', 'price': 6.2}
{'product': 'example5', 'price': 6.95}
{'product': 'example6', 'price': 7.7}
{'product': 'example7', 'price': 8.45}
{'product': 'example8', 'price': 9.2}
{'product': 'example9', 'price': 9.95}
{'product': 'example10', 'price': 10.7}
{'product': 'example11', 'price': 11.45}
{'product': 'example12', 'price': 12.2}
{'product': 'example13', 'price': 12.95}
{'product': 'example14', 'price': 13.7}
{'product': 'example15', 'price': 14.45}
{'product': 'example16', 'price': 15.2}
{'product': 'example17', 'price': 15.95}
{'product': 'example18', 'price': 16.7}
{'product': 'example19', 'price': 17.45}
```

In [21]:

```
aggregation_query = {
    "aggs": {
        "avg_price": {
            "avg": {
                "field": "price"
            }
        }
    }
}

# Execute the aggregation query
response = es.search(index=indexname, body=aggregation_query)

# Get the average price from the response
avg_price = response['aggregations']['avg_price']['value']

print(f"Average Price: {round(avg_price,2)}")
```

```
Average Price: 10.32

/var/folders/7l/srp8f0g91vxc5k32qtk6bwym0000gn/T/ipykernel_10552/228
62089.py:12: DeprecationWarning: The 'body' parameter is deprecated
and will be removed in a future version. Instead use individual para
meters.
  response = es.search(index=indexname, body=aggregation_query)
```

Removing all created indexes

In [22]:

```
es.indices.delete(index=indexname)
es.indices.delete(index=py_indexname)
```

Out[22]:

```
ObjectApiResponse({'acknowledged': True})
```

Thank you!

- I hope you found this notebook on ElasticSearch and Python helpful and insightful.
- If you have any questions, suggestions, or just want to connect, feel free to reach out to me on LinkedIn (https://www.linkedin.com/in/natalia-szczepanek/).

Let's continue the conversation, collaborate, and stay connected!

Happy coding!