OpenSearch Recipe Indexing Script

1. Introduction

This document contains a Python script designed for indexing recipe data into an OpenSearch cluster. The dataset is sourced from Kaggle, and the script follows these steps:

1. Load environment variables (e.g., OpenSearch connection details).
2. Connect to the OpenSearch cluster.
3. Define the index structure.
4. Load the dataset in JSON format.
5. Bulk index the recipes into OpenSearch.

Each part of the code is explained in detail below.

2. Environment Setup

The script uses the 'dotenv' library to load OpenSearch connection details from a '.env' file.

These variables include the host, port, username, and password required to connect to the OpenSearch cluster.

```
import json
from opensearchpy import OpenSearch, helpers
from dotenv import load_dotenv
import os

# Load environment variables from the .env file
load_dotenv()

# OpenSearch connection details from .env
host = os.getenv('OPENSEARCH_HOST')
port = int(os.getenv('OPENSEARCH_PORT'))
auth = (os.getenv('OPENSEARCH_USERNAME'), os.getenv('OPENSEARCH_PASSWORD'))
```

3. OpenSearch Client Initialization

The OpenSearch client is initialized using the connection details loaded from the environment variables. SSL options are configured to bypass hostname verification and suppress warnings.

```
# Initialize OpenSearch client
client = OpenSearch(
    hosts=[{'host': host, 'port': port}],
    http_compress=True,
    http_auth=auth,
    use_ssl=True,
    verify_certs=False,
    ssl_assert_hostname=False,
    ssl_show_warn=False,
)
```

4. Define Index Mapping

The index mapping specifies the structure of the data to be indexed, including fields like 'title', 'ingredients', 'categories', and nutritional information.

```
# Define index name
index_name = 'epirecipes'

# Define index mapping
mapping = {
    "settings": {
        "number_of_shards": 1,
        "number_of_replicas": 0
    },
    "mappings": {
        "properties": {
            "id": {"type": "integer"},
            "title": {"type": "text"},
            "ingredients": {"type": "text"},
            "categories": {"type": "keyword"},
            "calories": {"type": "integer"},
            "protein": {"type": "integer"},
            "fat": {"type": "integer"},
            "sodium": {"type": "integer"},
            "rating": {"type": "float"},
            "date": {"type": "date"},
            "desc": {"type": "text"},
            "directions": {"type": "text"}
        }
    }
}
```

5. Create Index

The script checks if the index already exists in OpenSearch. If not, it creates the index using the defined mapping.

```
# Create index if it doesn't exist
if not client.indices.exists(index=index_name):
    client.indices.create(index=index_name, body=mapping)
    print(f"Created index '{index_name}'")
else:
   print(f"Index '{index_name}' already exists")
```

6. Load and Index Data

The dataset is loaded from a JSON file and prepared for bulk indexing into OpenSearch. Each is indexed with fields such as 'title', 'ingredients', 'calories', and 'rating'.

```
# Load JSON data
file_path = '../data/epirecipes/full_format_recipes.json'
with open(file_path) as f:
    data = json.load(f)

# Prepare documents for bulk indexing
actions = []
for idx, doc in enumerate(data):
    actions.append({
        "_index": index_name,
        "_id": idx,  # Optionally use the row index or some unique ID
        "_source": {
            "id": idx,
            "title": doc.get('title', 'No Title'),
            "ingredients": doc.get('ingredients', []),
            "categories": doc.get('categories', []),
            "calories": doc.get('calories', 0),
            "protein": doc.get('protein', 0),
            "fat": doc.get('fat', 0),
            "sodium": doc.get('sodium', 0),
            "rating": doc.get('rating', 0.0),
            "date": doc.get('date', None),
            "desc": doc.get('desc', None),
            "directions": doc.get('directions', [])
        }
    })
```

7. Bulk Indexing

The documents are indexed in bulk using the 'helpers.bulk' method from the OpenSearch Python client.

```
# Bulk index the data into OpenSearch
try:
    helpers.bulk(client, actions)
    print(f"Indexed {len(actions)} documents into '{index_name}'")
except Exception as e:
    print(f"Error during bulk indexing: {e}")
```