In [ ]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import string
from pathlib import Path
from collections import Counter
import json
import nltk

nltk.download("punkt")
nltk.download("stopwords")

tf_idf = TfidfVectorizer(stop_words="english")


def encode_topics(df):
    # topics = df["topics"].str.get_dummies(sep=",")
    # topics = df["topics"].apply( topicfor topic in topics  )
    one_hot_encoded = (
        pd.get_dummies(df["topics"].apply(pd.Series).stack()).groupby(lev
    )
    df = pd.concat([df, one_hot_encoded], axis=1)
    # print(df)
    return df


def set_index(df, index_column="poll_ID"):
    df.set_index(index_column, inplace=True)
    return df


def reset_index(df):
    df.reset_index()
    return df


def check_column_type(df, column_name, check_type):
    column_index = df.columns.get_loc(column_name)
    for i in range(len(df)):
        if not isinstance(df.iloc[i, column_index], check_type):
            print(
                f"error: {df.iloc[i, 0], df.iloc[i, 1],df.iloc[i, 2], df.
            )


def preprocess_text(text):
    tokens = nltk.tokenize.word_tokenize(text)
    # tokens = [word.lower() for word in tokens if type(word) is str]
    tokens = [word.lower() for word in tokens]
    tokens = [word for word in tokens if word not in string.punctuation]
    stop_words = set(nltk.corpus.stopwords.words("english"))
    tokens = [word for word in tokens if word not in stop_words]
    processed_text = " ".join(tokens)

    return processed_text


def preprocess_list(field_list):
```

```
    ret_list = []
    stop_words = set(nltk.corpus.stopwords.words("english"))
    for item in field_list:
        tokens = nltk.tokenize.word_tokenize(item)
        # tokens = [word.lower() for word in tokens if type(word) is str
        tokens = [word.lower() for word in tokens]
        tokens = [word for word in tokens if word not in string.punctuati
        tokens = [word for word in tokens if word not in stop_words]
        processed_text = " ".join(tokens)
        ret_list.append(processed_text)

    return ret_list


def create_tf_idf_matrix(df, column):
    # print(f"{df[column]} is {df[column].dtype} and {df[column].dtype is
    df[column] = df[column].apply(lambda x: " ".join(x))
    df[column] = df[column].apply(preprocess_text)

    return tf_idf.fit_transform(df[column])


def create_souped_tf_idf_matrix(df):
    df["topics"] = df["topics"].apply(preprocess_list)
    df["question"] = df["question"].apply(preprocess_text)

    # Create a new soup feature
    df["soup"] = df.apply(create_soup, axis=1)

    return tf_idf.fit_transform(df["soup"])


def create_soup(df):
    res = (
        df["question"]
        + " "
        + " ".join(df["options"])
        + " "
        + (4 * (" " + " ".join(df["topics"])))
    )
    # print(f"----------------------------------\n* Processing: [{ }]")
    return res


def calc_cosine_similarity_matrix(tf_idf_matrix_1, tf_idf_matrix_2):
    return cosine_similarity(tf_idf_matrix_1, tf_idf_matrix_2)


def id_to_index(df, id):
    return df[df["id"] == id].index.values[0]


def title_from_idx(df, idx):
    return df[df.index == idx]


def gen_recommendations(
    index,
    df,
    cosine_similarity_matrix,
```

```
    number_of_recommendations,
):
    # index = idx_from_title(df, original_title)
    similarity_scores = list(enumerate(cosine_similarity_matrix[index]))
    similarity_scores_sorted = sorted(
        similarity_scores, key=lambda x: x[1], reverse=True
    )

    recommendations_indices = [
        t[0] for t in similarity_scores_sorted[1 : (number_of_recommendat
    ]
    recommendations = list(df["title"].iloc[recommendations_indices])
    # print(recommendations)
    # print(similarity_scores_sorted, type(similarity_scores_sorted))
    # recommendations_indices = [
    #     t[0] for t in similarity_scores_sorted[1 : (number_of_recommenda
    # ]
    # recommendations_scores = [
    #     t[1] for t in similarity_scores_sorted[1 : (number_of_recommenda
    # ]
    # return (df["title"].iloc[recommendations_indices], recommendations_

    return recommendations


def gen_rec_from_list_of_polls(
    interacted_polls, polls, cosine_similarity_matrix, number_of_recommen
):
    recommendations = []
    for poll_id in interacted_polls:
        index = id_to_index(polls, poll_id)
        similarity_scores = list(enumerate(cosine_similarity_matrix[index
        similarity_scores_sorted = sorted(
            similarity_scores, key=lambda x: x[1], reverse=True
        )

        recommendations_indices = [
            t[0] for t in similarity_scores_sorted[1 : (number_of_recomme
        ]
        recs = list(polls["id"].iloc[recommendations_indices])

        # Filter out polls that have already been interacted with
        filtered_recs = [poll for poll in recs if poll not in interacted_

        recommendations.append(filtered_recs)

    flattened_recommendations = [
        item for sublist in recommendations for item in sublist
    ]
    flattened_recommendations = Counter(flattened_recommendations)
    n_most_recommended = flattened_recommendations.most_common(
        number_of_recommendations
    )
    n_most_recommended = [t[0] for t in n_most_recommended]
    # print(n_most_recommended)

    return n_most_recommended
```

```
[nltk_data] Downloading package punkt to /home/erfan/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/erfan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [ ]:

```
from elasticsearch import Elasticsearch
import json
class ElasticsearchHandel:
    def __init__(self, elasticsearch_url, username, password, fingerprint
        self.elasticsearch_url = elasticsearch_url
        self.username = username
        self.password = password
        self.fingerprint = fingerprint
        self.client = Elasticsearch(
            hosts=self.elasticsearch_url,
            basic_auth=(self.username, self.password),
            ssl_assert_fingerprint=self.fingerprint,
        )

    def get_index(self, index_name, batch_size=100):
        setattr(self, index_name, [])
        index_list = getattr(self, index_name)
        from_index = 0
        all_instances = []

        while True:
            # query = {"query": {"match_all": {}}, "size": batch_size, "f
            results = self.client.search(
                index=index_name,
                query={"match_all": {}},
                size=batch_size,
                from_=from_index,
            )
            instances = results["hits"]["hits"]

            all_instances.extend(instances)
            from_index += batch_size
            if len(instances) < 100:
                break

        setattr(self, index_name, [instance["_source"] for instance in al
        return getattr(self, index_name)

    def get_interactions(self, index_name, user_id, batch_size=100):
        # setattr(self, index_name, [])
        # index_list = getattr(self, index_name)
        from_index = 0
        all_instances = []

        query = {
            "match_phrase": {"userId": user_id},
        }

        results = self.client.search(
            index=index_name,
            query=query,
            size=batch_size,
            from_=from_index,
            timeout="1s",
        )
```

```python
        # instances = results["hits"]["hits"][0]
        hits = results["hits"].get("hits")

        if not hits:
            # raise ValueError("User doesn't have any interactions.")
            raise InteractionNotFound()

        return hits[0].get("_source")

    def get_trend_polls(self, polls):
        # polls = getattr(self, "polls")
        # trend_polls = sorted(polls, key=lambda x: (-x["numberOfPollups'
        trend_polls = sorted(
            polls,
            key=lambda x: (
                -x["numberOfVotes"],
                -x["numberOfLike"],
                # -x["numberOfPollUp"],
            ),
        )

        # recs = trend_polls["id"]

        # print("\n", filtered_trend_polls, "\n")
        # setattr(self, "trend_polls", trend_polls)
        return trend_polls

    def export_index_to_file(self, index, index_file_path):
        try:
            with open(index_file_path, "w") as output:
                # for instance in self.instances:
                #        json.dump(instance["_source"], output, indent=4)
                json.dump(index, output, indent=4)
        except Exception as exp:
            print("Export Error", exp)
```

In [ ]:

```python
import pandas as pd

elasticsearch_url = "https://159.203.183.251:9200"
username = "pollett"
password = "9r0&rJP@19GY"
fingerprint = "CE:AA:F7:FF:04:C7:31:14:78:9C:62:D4:CE:98:F9:EF:56:DA:70:4


elastic_handle = ElasticsearchHandel(
    elasticsearch_url, username, password, fingerprint
)

polls = elastic_handle.get_index("polls")
trend_polls = elastic_handle.get_trend_polls(polls)

polls_df = pd.DataFrame.from_records(polls)

polls_df
```

Out[ ]:

	id	question	options	topics	pollType	ownerId	
0	016e4c36-84bf-48d1-9125-53fd65d3cec9	bbbb	[2, 1]	[General]	Public	61400ff7-531a-425e-a506-e2a900eec613	24T1
1	029cc519-fdbf-4f0a-8b38-803f2c1a2a4b	What is your favorite movie genre ?	[Romance, Science Fiction, Mystery/Thriller, H...	[Movies & TV shows]	Public	61400ff7-531a-425e-a506-e2a900eec613	17T0
2	03ec66cd-42fd-4de5-88e5-a97765238189	Test	[1, 2]	[General]	Public	e00b366a-37a8-407d-9a15-e585d1ad539a	10T1
3	058d2d5b-dc16-45de-a7c5-17cacfedd88d	aa	[2, 1]	[General]	Public	61400ff7-531a-425e-a506-e2a900eec613	24T1
4	0aa886f4-2891-4d1e-b6c0-695fb7ee6e3d	If you go back in time, would you change your ...	[Absolutely, No way]	[Science, General]	Public	e00b366a-37a8-407d-9a15-e585d1ad539a	26T0
...	...	...	...	...	...	...	
137	1ec1f108-2c7b-45d5-819c-9db42ff7a9ab	Which one ?	[one, two]	[Activity, Tech]	Public	66271a97-73ba-41c8-b460-23d166e4c020	27T0
138	5993e139-8a95-4c00-945b-1fa5400a7aee	Test	[1, 2]	[General]	Public	e00b366a-37a8-407d-9a15-e585d1ad539a	0001
139	4bd81737-c665-40a9-bbfd-827bc00c4443	Test	[1, 2]	[General]	Public	e00b366a-37a8-407d-9a15-e585d1ad539a	0001
140	ca327429-d945-45b0-9655-2aa060c813de	A	[1, 2]	[General]	Public	61400ff7-531a-425e-a506-e2a900eec613	0001
141	3e85fad9-7095-40ca-afda-40efb9be14d8	which one ?	[Samsung Galaxy S21 FE, Samsung Galaxy A54]	[Tech]	Public	67eb27ca-ba0b-4d29-8627-9ec78327b512	0001

142 rows × 12 columns

In [ ]:

```
polls_tf_idf_matrix = create_souped_tf_idf_matrix(polls_df)
polls_tf_idf_matrix
```

Out[ ]:

```
<142x274 sparse matrix of type '<class 'numpy.float64'>'
	with 586 stored elements in Compressed Sparse Row format>
```

The `polls_tf_idf_matrix` is a sparse matrix used to represent textual data in a numerical format. Let's break down its characteristics:

- **Dimensions**: The matrix has dimensions of 142 rows and 274 columns.
- **Sparse Matrix**: It's classified as a sparse matrix, meaning that the majority of its elements are zero. This is common in text data like TF-IDF matrices, where most terms do not appear in every document.
- **Data Type**: The elements of the matrix are of type `numpy.float64`, representing 64-bit floating-point numbers. This is the standard data type for TF-IDF values.
- **Stored Elements**: There are 586 non-zero elements (entries) in the matrix. Sparse matrices are memory-efficient because they only store these non-zero values.
- **Compressed Sparse Row Format (CSR)**: The matrix is stored in the Compressed Sparse Row (CSR) format, a widely used format for sparse matrices. It allows for efficient row-wise access and arithmetic operations.

In summary, the `polls_tf_idf_matrix` efficiently represents TF-IDF values of text data with 142 rows and 274 columns. Its sparse nature optimizes memory usage by storing only non-zero values, making it suitable for text analysis tasks.

In [ ]:
```
cosine_similarity_matrix = calc_cosine_similarity_matrix(
                polls_tf_idf_matrix, polls_tf_idf_matrix
            )
cosine_similarity_matrix
```

Out[ ]:
```
array([[1.        , 0.        , 0.7105551 , ..., 0.7105551 , 0.79536727,
        0.        ],
       [0.        , 1.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.7105551 , 0.        , 1.        , ..., 1.        , 0.89336728,
        0.        ],
       ...,
       [0.7105551 , 0.        , 1.        , ..., 1.        , 0.89336728,
        0.        ],
       [0.79536727, 0.        , 0.89336728, ..., 0.89336728, 1.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

Cosine Similarity Matrix Explanation:

- **Definition:** The `cosine_similarity_matrix` is a matrix designed to represent the similarity between pairs of polls within a dataset.
- **Calculation:** It is calculated using the `calc_cosine_similarity_matrix` function, which commonly utilizes the cosine similarity metric. Cosine similarity is a frequently used measure in natural language processing and information retrieval. It assesses the similarity between two vectors, in this context, the TF-IDF vectors representing the polls.

- **Interpretation:** In the `cosine_similarity_matrix`, each element `(i, j)` denotes the cosine similarity between two polls: poll `i` and poll `j`. The values within this matrix have a range from -1 to 1, and their meanings are as follows:
    - `1`: Indicates that the polls are identical or have the highest possible similarity.
    - `0`: Denotes that the polls are orthogonal, implying no similarity between them.
    - `-1`: Suggests that the polls are diametrically opposite or possess the highest possible dissimilarity.

This matrix is critical for generating recommendations as it quantifies the textual content's similarity or dissimilarity between different polls. By leveraging this similarity matrix, the recommendation system can identify polls with content similar to those the user has interacted with, resulting in more personalized and relevant recommendations.

In [ ]:
```
#user_id = request.args.get("userId")
user_id = "67eb27ca-ba0b-4d29-8627-9ec78327b512"



userInteractions = elastic_handle.get_interactions(
                "userpollinteractions", user_id
            )

userInteractions = [
                interaction["pollId"]
                for interaction in userInteractions["userPollActions"][:2
            ]
recommended_list = gen_rec_from_list_of_polls(
                userInteractions,
                polls_df,
                cosine_similarity_matrix,
                100,
            )

recommended_polls = polls_df[polls_df["id"].isin(recommended_list)]
recommended_polls
```

Out[ ]:

	id	question	options	topics	pollType	ownerId	
0	016e4c36-84bf-48d1-9125-53fd65d3cec9	bbbb	[2, 1]	[general]	Public	61400ff7-531a-425e-a506-e2a900eec613	24T11
1	029cc519-fdbf-4f0a-8b38-803f2c1a2a4b	favorite movie genre	[Romance, Science Fiction, Mystery/Thriller, H...	[movies tv shows]	Public	61400ff7-531a-425e-a506-e2a900eec613	17T06
2	03ec66cd-42fd-4de5-88e5-a97765238189	test	[1, 2]	[general]	Public	e00b366a-37a8-407d-9a15-e585d1ad539a	10T13
3	058d2d5b-dc16-45de-a7c5-17cacfedd88d	aa	[2, 1]	[general]	Public	61400ff7-531a-425e-a506-e2a900eec613	24T12
5	0c9f7ece-bf55-4193-b46d-ad8ab871246d	vote best tv show time	[Friends, The Office (US), Game of Thrones, Br...	[movies tv shows]	Public	61400ff7-531a-425e-a506-e2a900eec613	17T06
...	...	...	...	...	...	...	
123	c9e86380-41a2-4c66-b485-e2d978ff7711	studying english	[I'm thinking of studying in England., Because...	[politics, tech, sport]	Public	66271a97-73ba-41c8-b460-23d166e4c020	14T07
125	e749a6a3-592b-42be-a041-2eb155a9e95c	social media platoform prefer conduct poll	[Facebook, Instagram , LinkedIn, Pollett , Twi...	[tech, politics, science]	Private	08f0071c-397c-420d-a1fb-32f613a73398	14T15
128	a4e77a01-7e82-4bd7-8910-4333d01a96c4	string	[string, string]	[tech]	Public	67eb27ca-ba0b-4d29-8627-9ec78327b512	0001-
129	188c9313-a751-4211-85ab-14290e6c853d	string	[string, string]	[tech]	Public	67eb27ca-ba0b-4d29-8627-9ec78327b512	0001-
137	1ec1f108-2c7b-45d5-	one	[one, two]	[activity, tech]	Public	66271a97-73ba-41c8-	27T09

id	question	options	topics	pollType	ownerId
819c-9db42ff7a9ab					b460-23d166e4c020

100 rows × 13 columns