```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
```

```
import sklearn
print(sklearn.__version__)
```

```
1.5.2
```

```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-le
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-lea
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-le
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from sc
```

```
# Step 1: Load dataset
articles = pd.read_csv('/content/IMDB_Dataset.csv')
```

```
class NewsRecommender:
    def _init_(self, n_clusters=5):
        """
        Initialize the NewsRecommender with the number of clusters for KMeans.
        """
        self.n_clusters = n_clusters
        # Lower min_df to 1 to include words that appear in at least 1 document
        self.tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.8, min_df=1)
        self.kmeans = KMeans(n_clusters=self.n_clusters, random_state=42)
        self.article_vectors = None
        self.cluster_labels = None
        self.df = None



def preprocess(self):
    """
    Convert text content to TF-IDF vectors.
    """
    self.article_vectors = self.tfidf_vectorizer.fit_transform(self.df['content'])

def cluster_articles(self):
    """
    Cluster articles using KMeans and store the cluster labels.
    """
    self.cluster_labels = self.kmeans.fit_predict(self.article_vectors)
    self.df['cluster'] = self.cluster_labels
    def recommend(self, article_index, top_n=3):
        """
        Recommend similar articles based on clustering and cosine similarity.
```

```
        :param article_index: Index of the article for which recommendations are needed.
        :param top_n: Number of top recommendations to return.
        :return: DataFrame containing recommended articles with similarity scores.
        """
        if self.article_vectors is None or self.cluster_labels is None:
            raise RuntimeError("Model is not trained. Run preprocess() and cluster_articles() first.")

        article_cluster = self.df.loc[article_index, 'cluster']
        similar_articles = self.df[self.df['cluster'] == article_cluster]


   # Calculate cosine similarity
        article_vector = self.article_vectors[article_index]
        cluster_vectors = self.article_vectors[similar_articles.index]
        similarities = cosine_similarity(article_vector, cluster_vectors).flatten()



        # Combine with the cluster and rank by similarity
        similar_articles = similar_articles.copy()
        similar_articles['similarity'] = similarities
        recommendations = similar_articles.sort_values(by='similarity', ascending=False).head(top_n)

        return recommendations[['title', 'content', 'similarity']]



# Validate required columns
required_columns = ['Class Index', 'Title', 'Description']
if not all(col in articles.columns for col in required_columns):
    raise ValueError(f"Dataset must contain columns: {required_columns}")


# Handle missing values in the 'Description' column
# Replace NaN with an empty string
articles['Description'] = articles['Description'].fillna('')

# Step 2: TF-IDF vectorization
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
tfidf_matrix = tfidf_vectorizer.fit_transform(articles['Description'])


# Step 3: K-Means Clustering
num_clusters = 3  # Use 3 clusters
kmeans = KMeans(n_clusters=num_clusters, init='k-means++', random_state=42)
articles['kmeans_cluster'] = kmeans.fit_predict(tfidf_matrix)


# Step 4: Agglomerative Clustering
agglomerative = AgglomerativeClustering(n_clusters=num_clusters, linkage='ward')  # No 'affinity' argument
articles['agglomerative_cluster'] = agglomerative.fit_predict(tfidf_matrix.toarray())


# Step 5: DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5, metric='euclidean')  # Adjust `eps` and `min_samples` as needed
articles['dbscan_cluster'] = dbscan.fit_predict(tfidf_matrix.toarray())


# Step 6: Creating a Recommendation Function
def recommend_articles(user_history, articles_df, tfidf_matrix, top_n=5):
    user_tfidf = tfidf_matrix[articles_df['Class Index'].isin(user_history)]
    user_profile = np.asarray(user_tfidf.mean(axis=0)).reshape(-1)
```

```
    similarities = cosine_similarity(user_profile.reshape(1, -1), tfidf_matrix).flatten()
    recommendations = articles_df[~articles_df['Class Index'].isin(user_history)].copy()
    recommendations['similarity'] = similarities[~articles_df['Class Index'].isin(user_history)]
    return recommendations.sort_values(by='similarity', ascending=False).head(top_n)


# Display the recommended  user history
user_history = [1, 5, 20]
recommended_articles = recommend_articles(user_history, articles, tfidf_matrix)
print(recommended_articles[['Title', 'similarity']])
```

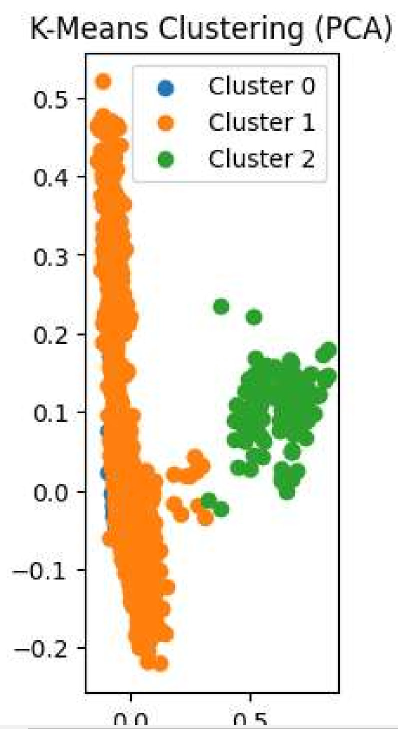
```
                                                  Title  similarity
177                                 Mission Accomplished!    0.282054
2030  Berlin Zoo Separates Baby Rhino from Clumsy Mo...    0.243830
12            Non-OPEC Nations Should Up Output-Purnomo    0.230048
173               Saudis: Bin Laden associate surrenders    0.225391
3729             NY Atty General Spitzer to Run for Gov.    0.217111
```

```
# Step 7: Visualization (PCA + Scatter Plot)
pca = PCA(n_components=2)
reduced_features = pca.fit_transform(tfidf_matrix.toarray())
plt.figure(figsize=(18, 6))
```

```
<Figure size 1800x600 with 0 Axes>
```

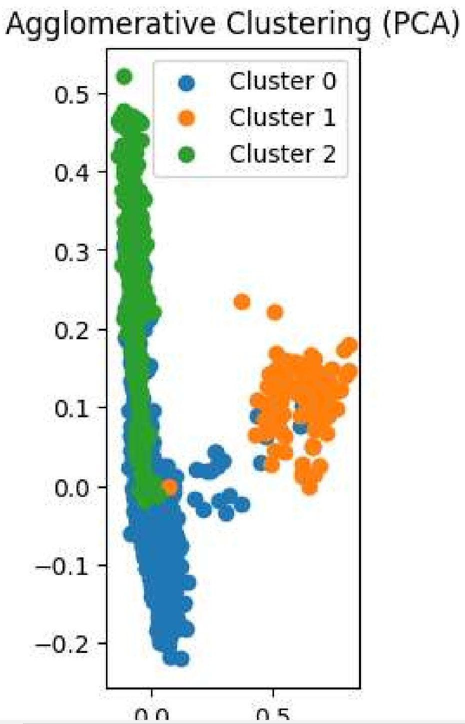
```
# Plot K-Means clusters
plt.subplot(1, 3, 1)
for cluster in range(num_clusters):
    cluster_points = reduced_features[articles['kmeans_cluster'] == cluster]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {cluster}')
plt.title('K-Means Clustering (PCA)')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x783fd57cf910>
```



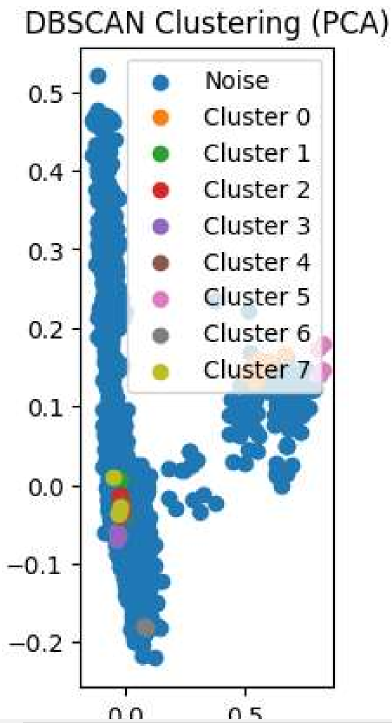
```
# Plot Agglomerative clusters
plt.subplot(1, 3, 2)
for cluster in range(num_clusters):
    cluster_points = reduced_features[articles['agglomerative_cluster'] == cluster]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {cluster}')
plt.title('Agglomerative Clustering (PCA)')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x783fd58322c0>
```




```
# Plot DBSCAN clusters
plt.subplot(1, 3, 3)
for cluster in np.unique(articles['dbscan_cluster']):
    cluster_points = reduced_features[articles['dbscan_cluster'] == cluster]
    label = f'Cluster {cluster}' if cluster != -1 else 'Noise'
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=label)
plt.title('DBSCAN Clustering (PCA)')
plt.legend()

plt.show()
```


DBSCAN Clustering (PCA)
Noise
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
0.0
0.5