به نام خدا

```python
import re
import nltk
import pandas as pd


def tokenize(text_array):
    for i in range(len(text_array)):
        if not pd.isna(text_array.loc[i]):
            text_array.loc[i] = text_array.loc[i].replace('\\n',' ')
            text_array.loc[i] = re.sub(r'^https?:\/\/.*[\r\n]*', '', text_array.loc[i], flags=re.MULTILINE)
            text_array.loc[i] = re.sub(r"[0-9]+", " ", text_array.loc[i])
            text_array.loc[i] = re.sub(r"[^a-zA-Z0-9]", " ", text_array.loc[i].lower())
            text_array.loc[i] = nltk.word_tokenize(text_array.loc[i])

    return text_array

data_csv = pd.read_csv('dataset/train.csv')

pure_tokens = tokenize(data_csv['Comment'])

pure_tokens_dataframe = pd.DataFrame(pure_tokens)
pure_tokens_dataframe.to_csv('./one_pure_tokens.csv', index=False)

```

در این تابع برای ستون کامنت لینک ها ، \n ها و کلماتی که با یک عدد شروع می‌شوند را جدا کردیم و سپس جملات را به کلمات توکنایز کردیم.

```python
import nltk
import pandas as pd


def stopword_remover(text_array):
    for i in range(len(text_array)):
        if not pd.isna(text_array.loc[i]):
            text_array.loc[i] = (text_array.loc[i]).replace("['","", 1).replace("']","", 1).split("', '")
    for i in range(len(text_array)):
        if type(text_array.loc[i]) != float:
            text_array.loc[i] = [word for word in text_array.loc[i] if word not in nltk.corpus.stopwords.words('english')]
    return text_array


pure_tokens_csv = pd.read_csv('step1/one_pure_tokens.csv')

tokens_without_stopwords = stopword_remover(pure_tokens_csv['Comment'])

tokens_without_stopwords_dataframe = pd.DataFrame(tokens_without_stopwords)
tokens_without_stopwords_dataframe.to_csv('./two_tokens_without_stopwords.csv', index=False)

```

در این تابع stop word ها را پاکسازی کردیم.

```python
import nltk
import pandas as pd


def stemming(text_array):
    for i in range(len(text_array)):
        if not pd.isna(text_array.loc[i]):
            text_array.loc[i] = (text_array.loc[i]).replace("['","", 1).replace("']","", 1).split("', '")
    for i in range(len(text_array)):
        if type(text_array.loc[i]) != float:
            text_array.loc[i] = [nltk.PorterStemmer().stem(w) for w in text_array.loc[i]]
    return text_array


tokens_without_stopwords_csv = pd.read_csv('step2/two_tokens_without_stopwords.csv')

tokens_with_stemming = stemming(tokens_without_stopwords_csv['Comment'])

tokens_with_stemming_dataframe = pd.DataFrame(tokens_with_stemming)
tokens_with_stemming_dataframe.to_csv('./three_tokens_with_stemming.csv', index=False)

```

در این تابع که استفاده نشده است ولی تست و بررسی شده است بر روی داده‌ها stemming انجام داده‌ایم ولی با توجه به نتیجه نچندان خوب از lemmetiziation استفاده کردیم.

```python
import pandas as pd
from nltk.stem import WordNetLemmatizer



def lemmatizing(text_array):
    for i in range(len(text_array)):
        if not pd.isna(text_array.loc[i]):
            text_array.loc[i] = (text_array.loc[i]).replace("['","", 1).replace("']","", 1).split("', '")
    for i in range(len(text_array)):
        if type(text_array.loc[i]) != float:
            text_array.loc[i] = [WordNetLemmatizer().lemmatize(w) for w in text_array.loc[i]]
    return text_array


tokens_without_stopwords_csv = pd.read_csv('step2/two_tokens_without_stopwords.csv')

tokens_with_lemmatizing = lemmatizing(tokens_without_stopwords_csv['Comment'])

tokens_with_lemmatizing_dataframe = pd.DataFrame(tokens_with_lemmatizing)
tokens_with_lemmatizing_dataframe.to_csv('./three_tokens_with_lemmatizing.csv', index=False)

```

در این تابع از lemmetizing برای پیدا کردن ریشه کلمات استفاده کردیم.

```python
import gensim
import pandas as pd


def word2Vec(text_array):
    for i in range(len(text_array)):
        if not pd.isna(text_array.loc[i]):
            text_array.loc[i] = (text_array.loc[i]).replace("['","", 1).replace("']","", 1).replace("', '", " ")
    text_array = text_array.apply(gensim.utils.simple_preprocess)
    # print(text_array)
    model = gensim.models.Word2Vec(window=3, min_count=3, workers=4)
    model.build_vocab(text_array, progress_per=10000)
    model.train(text_array, total_examples=model.corpus_count, epochs=70)
    model.save('word2vec.model')

    print(model.wv.get_vector('good'))

    # for i in range(len(text_array)):
    #     if type(text_array.loc[i]) != float:
    #         text_array.loc[i]
    #         print(str((i/13741)*100)+"%\r")
    return text_array


tokens_with_lemmatizing_csv = pd.read_csv('step3/three_tokens_with_lemmatizing.csv')

extracted_key_tokens = word2Vec(tokens_with_lemmatizing_csv['Comment'])

extracted_key_tokens_dataframe = pd.DataFrame(extracted_key_tokens)
extracted_key_tokens_dataframe.to_csv('./four_extracted_key_tokens.csv', index=False)
```

در این تابع از word۲vec آموزش داده شده مخصوص داده‌های train خودمان برای برداری کردن کلمات و سپس با گرفتن میانگین وزن دار کلمات هر داکیومنت برای برداری کردن داکیومنت ها استفاده کردیم. (این تابع به علت ضعیف‌تر بودن در برداری کردن داکیومنت ها کنار گذاشته شده و از تابع بعدی برای ادامه الگوریتم استفاده شده است)

```python
import gensim
import pandas as pd
from nltk.tokenize import word_tokenize


def doc2Vec(text_array):
    for i in range(len(text_array)):
        if not pd.isna(text_array.loc[i]):
            text_array.loc[i] = (text_array.loc[i]).replace("['","", 1).replace("']","", 1).replace("', '", " ")

    text_array = [gensim.models.doc2vec.TaggedDocument(words=word_tokenize(_d.lower()), tags=[str(i)]) for i, _d in enumerate(text_array)]
    documents_tags_dataframe = pd.DataFrame(text_array)
    documents_tags_dataframe.to_csv('./documents_tags.csv', index=False)

    model = gensim.models.Doc2Vec(dm=1, vector_size=65, hs=1, min_count=2, sample = 12000,window=3, alpha=0.025, min_alpha=0.00025)
    model.build_vocab(text_array)
    model.train(text_array, total_examples=model.corpus_count, epochs=70)
    model.save('doc2vec.model')

    return text_array


tokens_with_lemmatizing_csv = pd.read_csv('step3/three_tokens_with_lemmatizing.csv')

extracted_key_tokens = doc2Vec(tokens_with_lemmatizing_csv['Comment'])

extracted_key_tokens_dataframe = pd.DataFrame(extracted_key_tokens)
extracted_key_tokens_dataframe.to_csv('./four_point_one_extracted_key_tokens.csv', index=False)
```

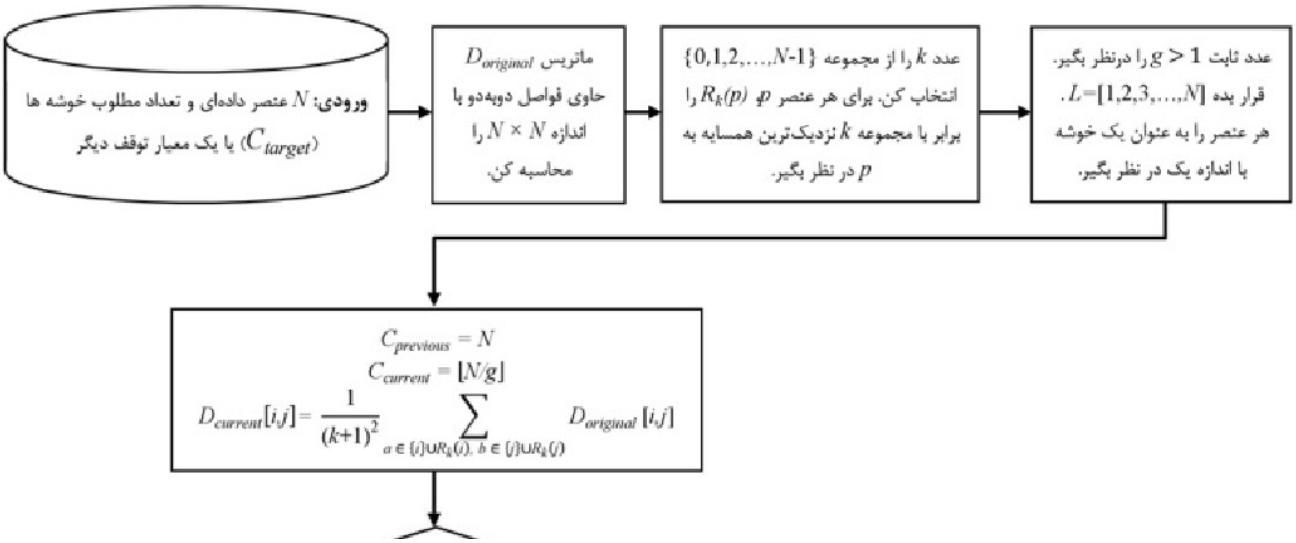
در این تابع با توجه به داکیومنت های موجود آن‌ها را با الگوریتم Doc۲vec ، train کردیم و بردار هایی به ازای هر مقاله ساخته شد.

```python
def save_function(original_cosine_distance, current_distance, indexes, document_cluster_labels, n_cluster_previous, n_cluster_current):
    np.save('./original_cosine_distance', original_cosine_distance)
    np.save('./current_distance', current_distance)
    np.save('./indexes', indexes)
    np.save('./first_current_distance_of_vectors', current_distance)

```

در این تابع مقادیر مرحله اول کلاسترینگ به علت زمان بر بودن و حجم زیاد ذخیره شدند تا در مرحله بعد این زمان از زمان اجرای ادامه کلاسترینگ کسر گردد.

```python
def key_identification_clustering_one(document_vectors, n_clusters):

    original_cosine_distance = np.zeros(shape=(8695, 8695))

    for i in range(len(document_vectors.distances(document_vectors[0]))):
        #calc cosine distances
        original_cosine_distance[i] = document_vectors.distances(document_vectors[i])

    original_cosine_distance = np.where(original_cosine_distance < 0, 0, original_cosine_distance)

    #calc nearest neighbors
    nearest_neighbors = NearestNeighbors(n_neighbors=k, metric="precomputed")
    nearest_neighbors.fit(original_cosine_distance)

    #Fetch kneighbors
    distances, indexes = nearest_neighbors.kneighbors()
    document_cluster_labels = [x for x in range(1, len(original_cosine_distance)+1)]
    n_cluster_previous = len(Counter(document_cluster_labels).keys())
    n_cluster_current = math.floor(n_cluster_previous/g)
    current_distance = np.zeros(shape=(n_cluster_previous, n_cluster_previous))

    for i in range(n_cluster_previous):
        for j in range(n_cluster_previous):
            current_distance_sigma = 0
            for l in range(k):
                for p in range(k):
                    current_distance_sigma += original_cosine_distance[indexes[i][l]][indexes[j][p]]

            current_distance[i][j] = current_distance_sigma/((k+1)**2)
        print("% ", ((i)/(n_cluster_previous))*100)
    print(current_distance)

    save_function(original_cosine_distance, current_distance, indexes, document_cluster_labels, n_cluster_previous, n_cluster_current)
    return document_vectors

k = 5
g = 40
n_cluster_target = 3

model = gensim.models.doc2vec.Doc2Vec.load("step4/doc2vec.model")

key_identification_clustering_one(model.dv, n_cluster_target)
```

در این تابع که قسمت اول کلاسترینگ را محاسبه می‌کند یعنی این قسمت از فلوچارت را :



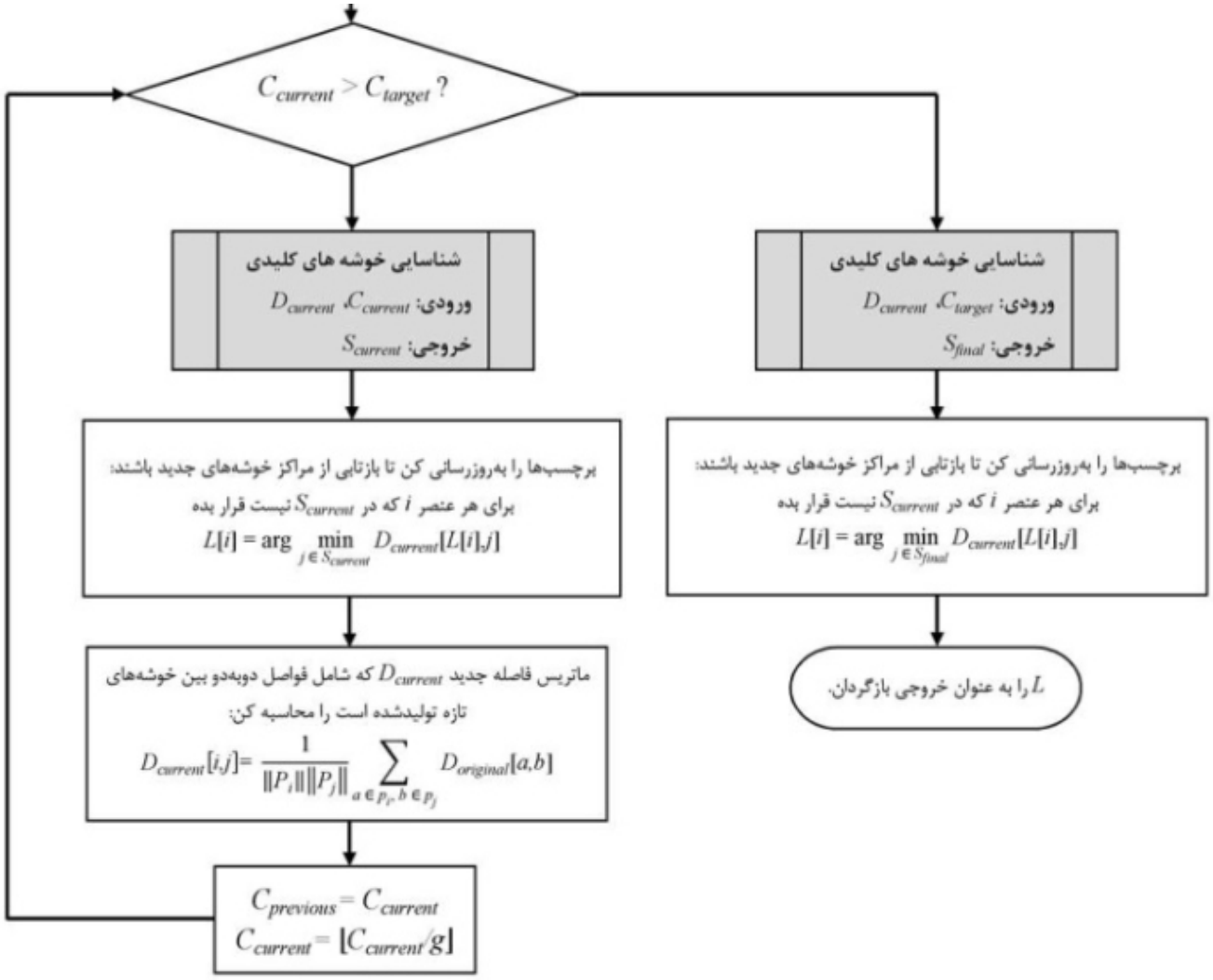
با استفاده از تابع فاصله کسینوسی فواصل بین داده‌ها و کلاستر ها محاسبه شده است.

```
14  def load_function():
15      original_cosine_distance = np.load('step5_1/original_cosine_distance.npy')
16      knn_indexes = np.load('step5_1/indexes.npy')
17      document_cluster_labels = [x for x in range(len(original_cosine_distance))]
18      n_cluster_previous = len(Counter(document_cluster_labels).keys())
19      n_cluster_current = math.floor(n_cluster_previous/g)
20      current_distance = np.load('step5_1/first_current_distance_of_vectors.npy')
21      return original_cosine_distance, current_distance, knn_indexes, document_cluster_labels, n_cluster_previous, n_cluster_current
```

در این تابع متغیر های ذخیره شده در قسمت اول الگوریتم load می شود.

```python
def key_item_selection(current_distance_matrix, n_clusters):

    mean_distance_matrix = np.mean(current_distance_matrix, axis=1)

    #s in article
    first_selected_key = np.argmin(mean_distance_matrix)
    selected_keys = []
    selected_keys.append(first_selected_key)

    #k in article
    unselected_items_for_key = [x for x in range(0, len(current_distance_matrix)) if x != first_selected_key]
    temp_min_key_j = -1
    temp_max_key_i = -1
    n = len(selected_keys)

    while n < n_clusters:
        temp_max_distance = 0
        for i in unselected_items_for_key:
            temp_min_distance = 1000000000
            for j in selected_keys:
                if current_distance_matrix[i][j] < temp_min_distance:
                    temp_min_distance = current_distance_matrix[i][j]
                    temp_min_key_j = j
            if current_distance_matrix[i][temp_min_key_j] > temp_max_distance:
                temp_max_distance = current_distance_matrix[i][temp_min_key_j]
                temp_max_key_i = i
        selected_keys.append(temp_max_key_i)
        unselected_items_for_key.remove(temp_max_key_i)
        n = n + 1
        # print("key_item_selection : %", ((n/n_clusters))*100)
    print(len(selected_keys))
    print(selected_keys)

    return selected_keys
```

در این تابع item های کلیدی طبق الگوریتم مقاله انتخاب می‌شوند برای شناخته شدن به عنوان مرکز کلاسترها



در این تابع که ادامه فایل قبل است این قسمت از فلوچارت در شکل زیر اجرا می شود

```
def key_identification_clustering_two(document_vectors, n_cluster_target):

    original_cosine_distance, current_distance, knn_indexes, document_cluster_labels, n_cluster_previous
, n_cluster_current = load_function()

    while n_cluster_current > n_cluster_target:

        selected_keys = key_item_selection(current_distance, n_cluster_current)

        # update elements lablels

        for i in range(len(document_cluster_labels)):
            min_distance = MAXINT
            for j in selected_keys:
                if current_distance[document_cluster_labels[i]][j] < min_distance:
                    min_distance = current_distance[document_cluster_labels[i]][j]
                    new_cluster_label = j
            document_cluster_labels[i] = new_cluster_label

        # update current distance matrix
        # p list in article
        counter = 0
        cluster_members_with_neighbors = {}
        for i in selected_keys:
            temp = list(locate(document_cluster_labels, lambda a: a == i))
            for l in temp:
                for j in range(k):
                    temp.append(knn_indexes[l][j])
                temp = list(set(temp))
            cluster_members_with_neighbors[counter] = temp
            counter += 1

        current_new_distance = np.zeros(shape=(n_cluster_current, n_cluster_current))
        for i in range(n_cluster_current):
            for j in range(n_cluster_current):
                current_distance_sigma = 0

                for l in cluster_members_with_neighbors[i]:
                    for p in cluster_members_with_neighbors[j]:
                        current_distance_sigma += original_cosine_distance[l][p]

                current_new_distance[i][j] = current_distance_sigma/(len(cluster_members_with_neighbors[
i])*len(cluster_members_with_neighbors[j]))

            print("update current distance matrix %", ((i)/(n_cluster_current))*100)

        # update n_cluster_previous & n_cluster_current
        n_cluster_previous = n_cluster_current
        n_cluster_current = math.floor(n_cluster_current/g)
        print("^^^")
        current_distance = current_new_distance

        c = 0
        cluster_label_dict_current = {}
        for i in selected_keys:
            cluster_label_dict_current[i] = c
            c += 1

        for i in range(len(document_cluster_labels)):
            document_cluster_labels[i] = cluster_label_dict_current[document_cluster_labels[i]]

    selected_keys = key_item_selection(current_distance, n_cluster_target)

    # update final elements lablels
    for i in range(len(document_cluster_labels)):
        min_distance = MAXINT
        for j in selected_keys:
            if current_distance[document_cluster_labels[i]][j] < min_distance:
                min_distance = current_distance[document_cluster_labels[i]][j]
                new_cluster_label = j
        document_cluster_labels[i] = new_cluster_label

    c = 0
    cluster_label_dict_current = {}
    for i in selected_keys:
        cluster_label_dict_current[i] = c
        c += 1

    for i in range(len(document_cluster_labels)):
        document_cluster_labels[i] = cluster_label_dict_current[document_cluster_labels[i]]

    return document_cluster_labels
```

در این تابع که طبق فلوچارت و توضیحات مقاله نوشته شده است الگوریتم مقاله اجرا می‌شود و در نهایت لیبل های پیش بینی شده برای داده‌های train مشخص می‌گردد.

در کد صفحه‌ی بعد دستورات مربوط به اجرای الگوریتم و گرفتن اطلاعاتی راجع به صحت و درصد درستی نتایج داده شده نوشته شده است و همچنین بازنمایی داکیومنت ها در اسکتر پلات برای درک بیشتر نمایش داده شده است.
و همچنین این الگوریتم همراه با الگوریتم خوشه بندی k-means اجرا شده تا تفاوت دقت این دو الگوریتم بررسی شود.

```
k = 3
g = 40
n_cluster_target = 3

model = gensim.models.doc2vec.Doc2Vec.load("step4/doc2vec.model")

labels = pd.read_csv('dataset/train.csv')

key_identification_clusters = key_identification_clustering_two(model.dv, n_cluster_target)

kmeans = KMeans(n_clusters=n_cluster_target)

li = np.zeros(shape=(8695, 65))
x = np.zeros(shape=(8695, 1))
y = np.zeros(shape=(8695, 1))

for i in range(len(model.dv)):
    li[i] = np.array(model.dv[i])
    x[i] = li[i][0]
    y[i] = li[i][1]

kmeans.fit(li)

#kmeans labels
# for i in range(len(kmeans.labels_)):
#     print(kmeans.labels_[i], labels["Topic"].iloc[i])

# Our labels
for i in range(len(key_identification_clusters)):
    print(key_identification_clusters[i], labels["Topic"].iloc[i])

Biology_0 = 0
Biology_1 = 0
Biology_2 = 0

Chemistry_0 = 0
Chemistry_1 = 0
Chemistry_2 = 0

Physics_0 = 0
Physics_1 = 0
Physics_2 = 0

for i in range(len(key_identification_clusters)):
    if key_identification_clusters[i] == 0:

        if labels["Topic"].iloc[i] == "Biology":
            Biology_0 += 1
        if labels["Topic"].iloc[i] == "Chemistry":
            Chemistry_0 += 1
        if labels["Topic"].iloc[i] == "Physics":
            Physics_0 += 1

    if key_identification_clusters[i] == 1:

        if labels["Topic"].iloc[i] == "Biology":
            Biology_1 += 1
        if labels["Topic"].iloc[i] == "Chemistry":
            Chemistry_1 += 1
        if labels["Topic"].iloc[i] == "Physics":
            Physics_1 += 1

    if key_identification_clusters[i] == 2:

        if labels["Topic"].iloc[i] == "Biology":
            Biology_2 += 1
        if labels["Topic"].iloc[i] == "Chemistry":
            Chemistry_2 += 1
        if labels["Topic"].iloc[i] == "Physics":
            Physics_2 += 1


#kmeans colors
kmeans_colors = kmeans.labels_

#key_identification_cluster colors
key_identification_cluster_colors = np.asarray(key_identification_clusters)

#main colors
main_colors = np.zeros(shape=(8695, 1))

for i in range(len(labels["Topic"])):
    if labels["Topic"].iloc[i] == "Biology":
        main_colors[i] = 0
    if labels["Topic"].iloc[i] == "Chemistry":
        main_colors[i] = 1
    if labels["Topic"].iloc[i] == "Physics":
        main_colors[i] = 2


print("Kmeans Accuracy:", (100*metrics.accuracy_score(main_colors, kmeans_colors)))
print("Key Identification Accuracy:", (100*metrics.accuracy_score(main_colors,
key_identification_cluster_colors)))

print("Biology_0", Biology_0)
print("Biology_1", Biology_1)
print("Biology_2", Biology_2)
print("Chemistry_0", Chemistry_0)
print("Chemistry_1", Chemistry_1)
print("Chemistry_2", Chemistry_2)
print("Physics_0", Physics_0)
print("Physics_1", Physics_1)
print("Physics_2", Physics_2)

plt.scatter(x, y, c=kmeans_colors)
plt.show()
plt.scatter(x, y, c=key_identification_cluster_colors)
plt.show()
plt.scatter(x, y, c=main_colors)
plt.show()
```