KDAG GENRE ANALYSIS

USING PANDAS, NUMPY AND MATPLOTLIB AS SUGGESTED

We are taking input using pd.DataFrame method for importing the data

I defined functions so as to apply even in the later stage

- Bow =
- bow is same for all the present keywords in an item
- Like its either 1/0

- **Tfidf = tf is same as bow and then finding the inverse document frequency**

  Idf = we want to find out the no of times the key has been repeated amount the entire dataset

  Idf = log[total no of items / no of items in which key is present]

  But to avoid errors I used standard formula like in sklearn

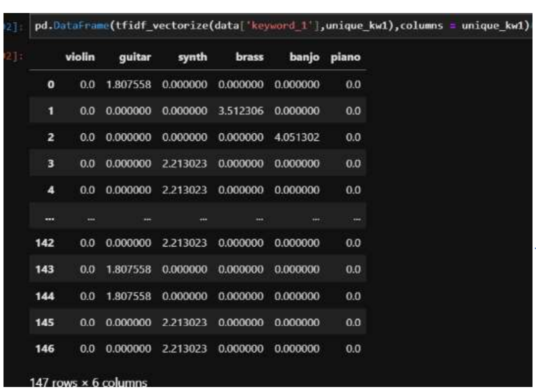
  Idf = log[1+total no of items / 1+ no of items in which key is present] +1

  [by LOGIC it should have been counted by taking idf on each genres dataset separately ,

  For machine learning outputs , but didn't as it was mentioned not to use genres in the analyzing part]

- **standardize_data**: Standardizes the data to have a mean of 0 and a standard deviation of 1.

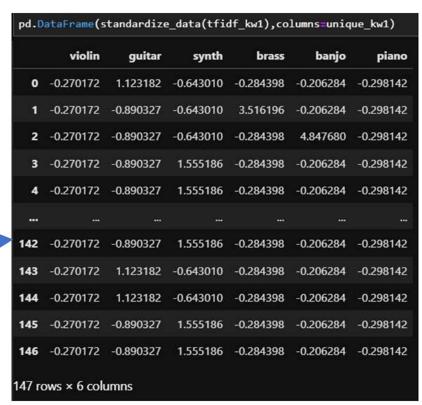
  The standardizing part involves bringing the mean of all points to the center and then scaling it , by doing std deviation to 1



```
pd.DataFrame(tfidf_vectorize(data['keyword_1'],unique_kw1),columns = unique_kw1)
```

	violin	guitar	synth	brass	banjo	piano
0	0.0	1.807558	0.000000	0.000000	0.000000	0.0
1	0.0	0.000000	0.000000	3.512306	0.000000	0.0
2	0.0	0.000000	0.000000	0.000000	4.051302	0.0
3	0.0	0.000000	2.213023	0.000000	0.000000	0.0
4	0.0	0.000000	2.213023	0.000000	0.000000	0.0
...	...	...	...	...	...	...
142	0.0	0.000000	2.213023	0.000000	0.000000	0.0
143	0.0	1.807558	0.000000	0.000000	0.000000	0.0
144	0.0	1.807558	0.000000	0.000000	0.000000	0.0
145	0.0	0.000000	2.213023	0.000000	0.000000	0.0
146	0.0	0.000000	2.213023	0.000000	0.000000	0.0

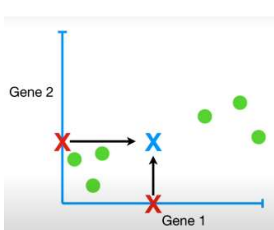
147 rows × 6 columns

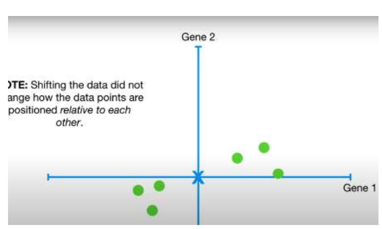


```
pd.DataFrame(standardize_data(tfidf_kw1),columns=unique_kw1)
```

	violin	guitar	synth	brass	banjo	piano
0	-0.270172	1.123182	-0.643010	-0.284398	-0.206284	-0.298142
1	-0.270172	-0.890327	-0.643010	3.516196	-0.206284	-0.298142
2	-0.270172	-0.890327	-0.643010	-0.284398	4.847680	-0.298142
3	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
4	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
...	...	...	...	...	...	...
142	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
143	-0.270172	1.123182	-0.643010	-0.284398	-0.206284	-0.298142
144	-0.270172	1.123182	-0.643010	-0.284398	-0.206284	-0.298142
145	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
146	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142

147 rows × 6 columns





KDAG GENRE ANALYSIS

- **pca_decomp**:

```
def pca_decomp(data):
    cov_array = np.cov(standardize_data(data), rowvar=False) #signifies the relation between how two keywords depend upon on each other
    eigenvectors = np.linalg.eig(cov_array)[1]

    df1 = pd.DataFrame(np.linalg.eig(cov_array)[0],columns = ['eigenvalues'])
    df2 = pd.DataFrame(eigenvectors)
    df = pd.concat([df1,df2],axis=1)
    df.sort_values(by='eigenvalues',ascending=False,inplace=True)
    pca_result = np.dot(standardize_data(data), df.iloc[:,1:].to_numpy()[:, :2])
    return pca_result
```

I used taking dataframes eigen vectors and eigen values so as to sort , there was no other use of df here

I will find for some other method too which is possible

cov matrix signifies the relation between how two keywords depend upon on each other

```
[113]: cov_array_kw1 = np.cov(standardize_data(tfidf_kw1), rowvar=False)
       pd.DataFrame(cov_array_kw1,columns = unique_kw1,index=unique_kw1)
```

[113]:

	violin	guitar	synth	brass	banjo	piano
violin	1.006849	-0.242189	-0.174913	-0.077363	-0.056114	-0.081101
guitar	-0.242189	1.006849	-0.576410	-0.254942	-0.184918	-0.267262
synth	-0.174913	-0.576410	1.006849	-0.184123	-0.133551	-0.193022
brass	-0.077363	-0.254942	-0.184123	1.006849	-0.059069	-0.085372
banjo	-0.056114	-0.184918	-0.133551	-0.059069	1.006849	-0.061923
piano	-0.081101	-0.267262	-0.193022	-0.085372	-0.061923	1.006849

and then eigenvectors and eigenvalues are basic significance of eigen value becomes the magnitude of that point in the resembled cov matrix

and eigen vector signifies the x and y components of the points

```
[124]: pd.DataFrame(eigenvectors)
```

[124]:

	0	1	2	3	4	5
0	0.300873	0.090184	0.382697	-0.237514	-0.809768	-0.206755
1	0.593454	-0.755961	-0.274497	-0.026789	0.011433	0.011776
2	0.543593	0.629543	-0.553161	-0.039396	0.017537	0.018331
3	0.314405	0.096405	0.429169	-0.190953	0.543022	-0.613460
4	0.236433	0.065026	0.236291	0.937434	-0.056648	-0.045241
5	0.327175	0.102680	0.481859	-0.161469	0.213905	0.760524

```
[122]:  eigenvalues, eigenvectors = np.linalg.eig(cov_array)
        eigenvalues

[122]:  array([2.22044605e-16, 1.60047918e+00, 1.20986813e+00, 1.05466186e+00,
               1.08343320e+00, 1.09265352e+00])
```

```
pd.DataFrame(standardize_data(tfidf_kw1),columns=unique_kw1)
```

	violin	guitar	synth	brass	banjo	piano
0	-0.270172	1.123182	-0.643010	-0.284398	-0.206284	-0.298142
1	-0.270172	-0.890327	-0.643010	3.516196	-0.206284	-0.298142
2	-0.270172	-0.890327	-0.643010	-0.284398	4.847680	-0.298142
3	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
4	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
...	...	...	...	...	...	...
142	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
143	-0.270172	1.123182	-0.643010	-0.284398	-0.206284	-0.298142
144	-0.270172	1.123182	-0.643010	-0.284398	-0.206284	-0.298142
145	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142
146	-0.270172	-0.890327	1.555186	-0.284398	-0.206284	-0.298142

147 rows × 6 columns

KDAG GENRE ANALYSIS

- **plot pca :**

  we sort the eigenvectors based upon eigen values

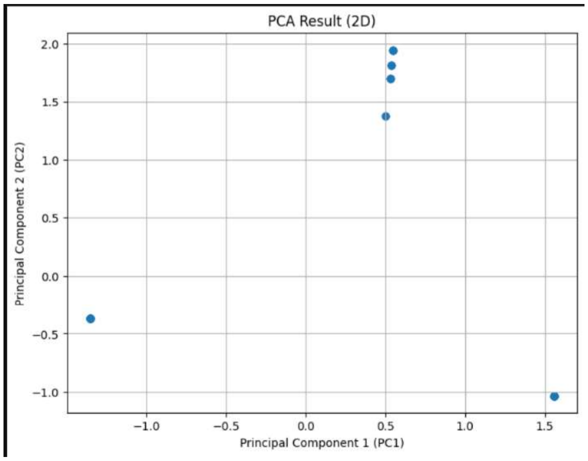
  we use the first two eigen vectors as the base of the two axis and plot them creating a total resemblance in 2d ,
  if wanted in 3d would have done first three

WHEN I PERFORMED THE PCA FOR JUST THE KEYWORD 1

THE GRAPH REPRESENTS HOW the PCA did its job of converting the to a plot it in 2D graph

As there were 6 unique words in keyword 1
we have 6 points here on pca

PCA does it by retaining the important / relevant data ie its variance



kmeans: implemented the k means algorithm.

I couldn't use the elbow method , as I wasn't able to clearly break it down by myself

But I tried on using AI and got the response that 4 clusters are best

(basic logic I understood that we want to find diff silhouette score and then find the k for which s score is best)

```
k=4 #define the number of clusters
def kmeans(data, k= k, max_iterations=100):
    centroids = data[np.random.choice(data.shape[0], k, replace=False)]

    for _ in range(max_iterations):
        distances = np.sqrt(((data - centroids[:, np.newaxis])**2).sum(axis=2))
        labels = np.argmin(distances, axis=0)
        new_centroids = np.array([data[labels == i].mean(axis=0) for i in range(k)])

        # If the centroids don't change
        if np.all(centroids == new_centroids):
            break
        # centroids = new_centroids
    return labels, new_centroids, centroids
```

On 1st iteration it counts the centroids randomly and then after that it changes the centroids with the average location from the mean of all the data points in that cluster

So on it carries out 100 operations and then returns the final 100th label

KDAG GENRE ANALYSIS

I didn't know how to implement
taking maximum amount of times when one point is allotted certain label , you a lot that label to that point , but I will soon perform it

Silhouette Score Calculation:

Calculated the silhouette score by using the approach

That if the point is more closer to the points in its cluster than the others cluster then it's a better clustering done

So that we are doing for each combination of point and then averaging it out at the end to create the silhouette score

I have obtained a silhouette score around 66-73% in general most of the times

It changes as the initial point of the centroids had changed so correspondingly final cluster changed so for different labelling , different silhouette score is obtained

Here I have used the genre provided to visualize or intuitive thinking of checking each step's accuracy

AT last I have also performed some analysis for finding out the accuracy of my code and find some implementation techniques from all the analysis done

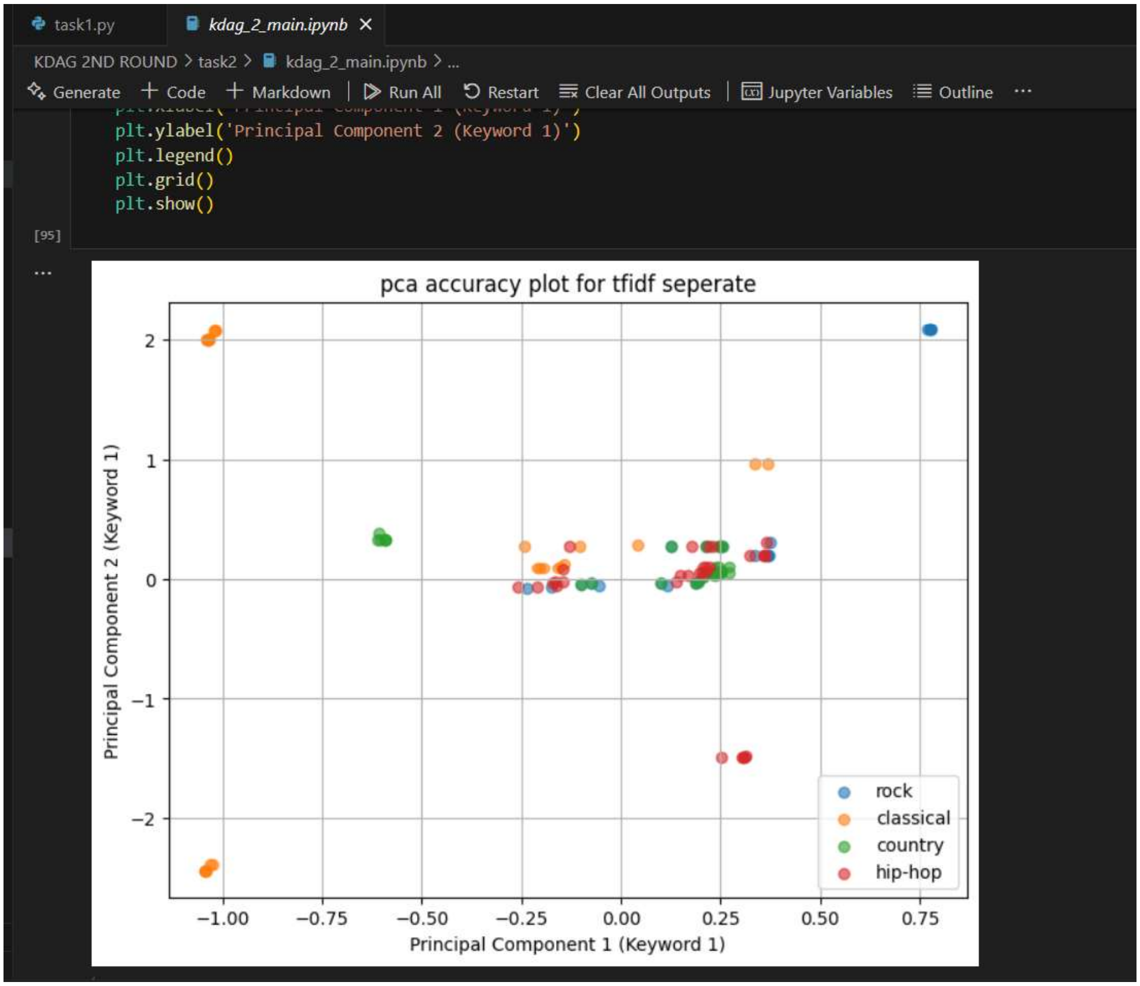
Like for finding out the percentage of each genre present in my clusters

```
#for seperated keywords with tfidf
# Calculate the percentage distribution of genres in each cluster
tfidf_test = k_tfidf_sep.copy()
tfidf_test["genre"] = data["genre"]
cluster_genre_distribution = tfidf_test.groupby(['cluster','genre']).size().unstack(fill_value=0)

# Normalize the distribution to percentages
tfidf_sep_truth = cluster_genre_distribution.div(cluster_genre_distribution.sum(axis=1), axis=0) * 100

# Display the genre distribution in each cluster
tfidf_sep_truth
```

genre	classical	country	hip-hop	pop	rock
cluster					
0	47.058824	15.686275	15.686275	11.764706	9.803922
1	5.714286	14.285714	22.857143	17.142857	40.000000
2	1.851852	22.222222	25.925926	29.629630	20.370370
3	0.000000	57.142857	0.000000	42.857143	0.000000

KDAG GENRE ANALYSIS

Similarly for finding out the pca and tfidf accuracy I plotted



For this I just used the same code for plotting the cluster just the labels provided by me using genre

```
# ground truth column addition to the data
true_clusters = np.zeros(len(data))

unique_genre = (data['genre']).unique()

for i,word in enumerate(data["genre"]):
    single=word.split()
    for genre in single:
        if genre == unique_genre[0]:
            true_clusters[i] = 0
        elif genre == unique_genre[1]:
            true_clusters[i] = 1
        elif genre == unique_genre[2]:
            true_clusters[i] = 2
        elif genre == unique_genre[3]:
            true_clusters[i] = 3
        elif genre == unique_genre[4]:
            true_clusters[i] = 4

data['true_clusters'] = true_clusters
data
```
[94]

KDAG GENRE ANALYSIS

For using it to output something

```
def weight(kw_inp, word_column ,data = data.copy()):
        kw_weightage = np.zeros(len(data))
        for i,word in enumerate(word_column):
            single =word.split()
            for keyword in single:
                if keyword == kw_inp:
                    kw_weightage[i] += 1
        return kw_weightage

def weightage(kw1_inp,kw2_inp,kw3_inp,data1=data.copy()):
        kw1_weightage = weight(kw1_inp,data1['keyword_1'])
        kw2_weightage = weight(kw2_inp,data1['keyword_2'])
        kw3_weightage = weight(kw3_inp,data1['keyword_3'])
        weightage =((kw1_weightage) + (kw2_weightage) + (kw3_weightage)
        return weightage

def genre_predictor(weightage,soham,data1=data.copy()):
        pc1_expected = 0
        pc2_expected = 0
        data1['weightage'] = weightage
        for i, value in enumerate(soham['PC1']*data1['weightage']):
                pc1_expected +=value
        pc1_expected = pc1_expected/len(data1)
        for i , value in enumerate(soham['PC2']*data1['weightage']):
                pc2_expected +=value
        pc2_expected = pc2_expected/len(data1)
        combine = {
              'PC1': [pc1_expected],
              'PC2': [pc2_expected]}
        combine_df = pd.DataFrame(combine)
        soham = pd.concat([soham, combine_df], ignore_index=True)
        labels =kmeans(soham.to_numpy())[0]
        soham['cluster'] = labels
```

KDAG GENRE ANALYSIS

Printing the output with the best silhouette score and then finding out the highest percentage of the genre in that cluster of that type of vectorization

```python
kw1_inp = 'guitar' #input("enter 1st keyword")
kw2_inp = 'happy'  #input("enter 2nd keyword")
kw3_inp = 'upbeat'  #input("enter 3rd keyword")

bank = ['classical', 'country', 'hip-hop', 'pop', 'rock']
#answer = country

scores=[score_tf,score_bow]
max_score = max(scores)
print(f'max_score = {max_score}')
if max_score == score_tf:
    soham = pd.DataFrame(tfidf_pca_array, columns=['PC1', 'PC2'])
    results = genre_predictor(weightage(kw1_inp,kw2_inp,kw3_inp),soham)
    print(f'TF-IDF,Cluster = {results}')
    row = tfidf_sep_truth.loc[results]
    print(row)
    max_per = max(row)
    max_percent_ind = [i for i, som1 in enumerate(row) if som1 == max_per]
    expected = []
    for index in max_percent_ind:
        expected.append(bank[index])
    print(f"Expected Genres: {expected}")


if max_score == score_bow :
    soham = pd.DataFrame(bow_pca_array, columns=['PC1', 'PC2'])
    results = genre_predictor(weightage(kw1_inp,kw2_inp,kw3_inp),soham)
    print(f'BoW,Cluster = {results}')
    row = bow_sep_truth.loc[results]
    print(row)
    max_per = max(row)
    max_percent_ind = [i for i, som1 in enumerate(row) if som1 == max_per]
    expected = []
```

We can also interpret from this that the tf-idf results better in most of the cases

note that while combining I addressed 0.6 to keyword 1 while 0.2 each to the other two

Because in analyzing humanly , I thought that the genres most related to keyword 1 on analyzing the keys separately for single genre humanly

And I took cube root of the cubes of the pca while combining by trial and errors

KDAG GENRE ANALYSIS

Generally my accuracy ranged from 18- 38%

```python
def accuracy(data1 = data.copy()):
    correct = 0
    for i in range(len(data1)):
        weightage1 = weightage(data1['keyword_1'][i],data1['keyword_2'][i],data1['keyword_3'][i])
        prediction = genre_predictor(weightage1,k_tfidf_sep)
        row = tfidf_sep_truth.loc[prediction]
        max_per = max(row)
        max_percent_ind = [i for i, som1 in enumerate(row) if som1 == max_per]
        expected = []
        for index in max_percent_ind:
            expected.append(bank[index])
        for word in expected:
            if word == data1['genre'][i]:
                correct += 1

    return correct/len(data1)*100
```
[98]