TEAM 1

Reporter： Hanqing Liu

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn import svm
```

1. about Dataset

It's part of the data provided in the ICM test in 2021, containing the information about songs in the past 100 years.

Each row represents one song, while the columns decribe the basic information about the song.

The detail information about the columns goes as follow:

In [2]:
```
latin=pd.read_csv('D:/美赛2021/D/2021_ICM_Problem_D_Data/按流派（歌）/latin.csv')
poprock=pd.read_csv('D:/美赛2021/D/2021_ICM_Problem_D_Data/按流派（歌）/pop or rock.cs
classical=pd.read_csv('D:/美赛2021/D/2021_ICM_Problem_D_Data/按流派（歌）/classical.cs
avantgarde=pd.read_csv('D:/美赛2021/D/2021_ICM_Problem_D_Data/按流派（歌）/Avant-Gard
electronic=pd.read_csv('D:/美赛2021/D/2021_ICM_Problem_D_Data/按流派（歌）/Electronic.
avantgarde.head()
```

Out[2]:

	Unnamed: 0	Unnamed: 0.1	artist_names	artists_id	danceability	energy	valence	tempo	loudness
0	4356	4356	['Moondog']	[496525]	0.542	0.378	0.609	0.6	0.687450
1	4357	4357	['Moondog']	[496525]	0.657	0.373	0.975	0.2	0.692583
2	4358	4358	['Moondog']	[496525]	0.653	0.988	0.284	0.6	0.704333
3	4359	4359	['Moondog']	[496525]	0.437	0.499	0.643	0.6	0.780000
4	4360	4360	['Moondog']	[496525]	0.446	0.525	0.193	1.0	0.685667

5 rows × 22 columns

- danceability: A measure of how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable. (float)
- energy: A measure representing a perception of intensity and activity. A value of 0.0 is least intense/energetic and 1.0 is most intense/energetic. Typically, energetic tracks feel fast,

loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy. (float)

- valence: A measure describing the musical positiveness conveyed by a track. A value of 0.0 is most negative and 1.0 is most positive. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry). (float)
- tempo: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration. (float)
- loudness: The overall loudness of a track in decibels (dB). Values typical range between -60 and 0 db. Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). (float)
- mode: An indication of modality (major or minor), the type of scale from which its melodic content is derived, of a track. Major is represented by 1 and minor is 0.
- key: The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value for key is -1. (integer)
- acousticness: A confidence measure of whether the track is acoustic (without technology enhancements or electrical amplification). A value of 1.0 represents high confidence the track is acoustic. (float)
- instrumentalness: Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0. (float)
- liveness: Detects the presence of an audience in a track. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live. (float)
- speechiness: Detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks. (float) - explicit: Detects explicit lyrics in a track (true (1) = yes it does; false (0) = no it does not OR unknown). (Boolean)
- duration_ms: The duration of the track in milliseconds. (integer)
- popularity: The popularity of the track. The value will be between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Generally speaking, songs that are being played more frequently now will have a higher popularity than songs that were played more frequently in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity are derived mathematically from track popularity. (integer)
- year: The year of release of a track. (integer from 1921 to 2020)

2. Data Pre-proccesing

- Missing values, blank values and illegal values are found through observation and functions such as np.where, and the data is relatively clean and does not need to be processed.
- The loudness data range is (-60, 0), and in keeping with most of the other data, I used loudness=(loudness+60)/60 to limit it to (0,1).
- According to music theory, it is unscientific to take the duration of music as an important factor to measure music. For example, it is difficult to admit that there is a significant difference between a 3 minute music and a 3 minute and 5 second music.The same is true for rhythm.So divide them into several intervals, and assign the same value to the data in the same interval, from 0 to 1.
- The popularity range is (0,100), and I use popularity=popularity/100 to limit it to (0,1).
- The key range is (0,11), and I use key=key/11 to limit it to (0,1).

now all the elements goes between 0 and 1, and at the same time are properly distributed.

3. Problems to Solve

- Clustering: The study of whether a certain number of detailed classifications exist within a musical genre
- Regression: Study the tendency of a particular attribute of a single artist's work over time
- Categorization: Search for criteria to distinguish two types of songs based on their attributes

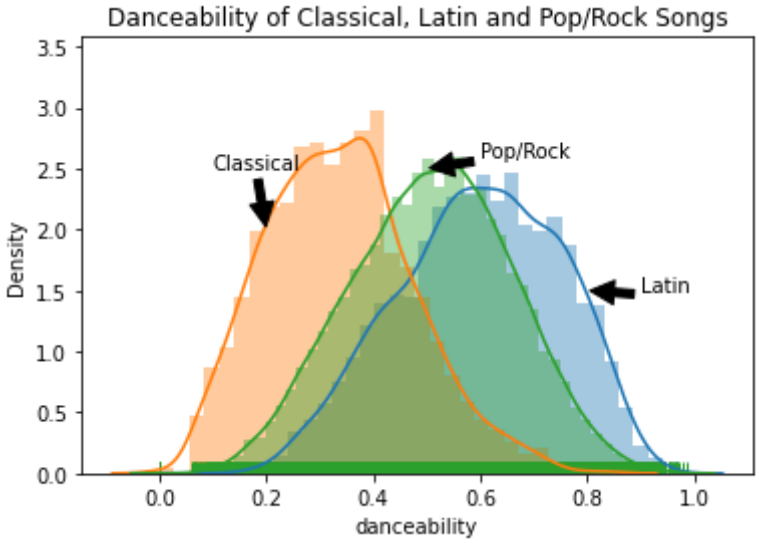
4. Visualization

4.1 compare the same attribute between different genre

In [3]:
```
dan = sns.distplot(latin['danceability'], rug=True)
dan.set_title('Danceability of Classical, Latin and Pop/Rock Songs')
dan = sns.distplot(classical['danceability'], rug=True)
dan= sns.distplot(poprock['danceability'], rug=True)
dan.annotate('Latin', xy=(0.8, 1.5), xytext=(0.9, 1.5), arrowprops=dict(facecolor='bl
dan.annotate('Pop/Rock', xy=(0.5, 2.5), xytext=(0.6, 2.6), arrowprops=dict(facecolor=
dan.annotate('Classical', xy=(0.2, 2.0), xytext=(0.1, 2.5), arrowprops=dict(facecolor
print('average danceability of latin songs is '+str(np.mean(latin['danceability'])))
print('average danceability of classical songs is '+str(np.mean(classical['danceabili
print('average danceability of pop/rock songs is '+str(np.mean(poprock['danceability'
```

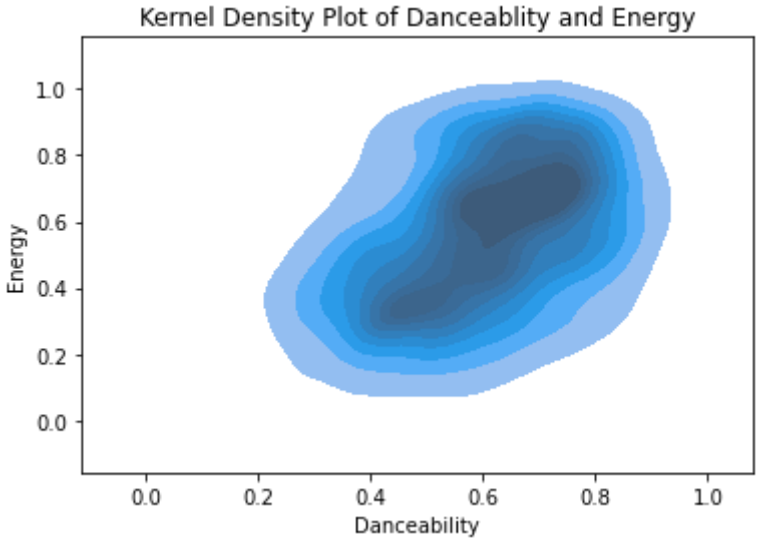
```
D:\INSTALL\Ana\envs\py38\lib\site-packages\seaborn\distributions.py:2551: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\INSTALL\Ana\envs\py38\lib\site-packages\seaborn\distributions.py:2055: FutureWarnin
g: The `axis` variable is no longer used and will be removed. Instead, assign variable
s directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
D:\INSTALL\Ana\envs\py38\lib\site-packages\seaborn\distributions.py:2551: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\INSTALL\Ana\envs\py38\lib\site-packages\seaborn\distributions.py:2055: FutureWarnin
g: The `axis` variable is no longer used and will be removed. Instead, assign variable
```

```
s directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
D:\INSTALL\Ana\envs\py38\lib\site-packages\seaborn\distributions.py:2551: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
D:\INSTALL\Ana\envs\py38\lib\site-packages\seaborn\distributions.py:2055: FutureWarnin
g: The `axis` variable is no longer used and will be removed. Instead, assign variable
s directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
average danceability of latin songs is 0.5969784287011807
average danceability of classical songs is 0.3366323688969259
average danceability of pop/rock songs is 0.5090829181876744
```



4.2 the potential relationship between danceability and energy

In [4]:
```
ax = sns.kdeplot(x=latin['danceability'],
                 y=latin['energy'],
                 shade=True) # shade will fill in the contours
ax.set_title('Kernel Density Plot of Danceablity and Energy')
ax.set_xlabel('Danceability')
ax.set_ylabel('Energy')
plt.show()
```

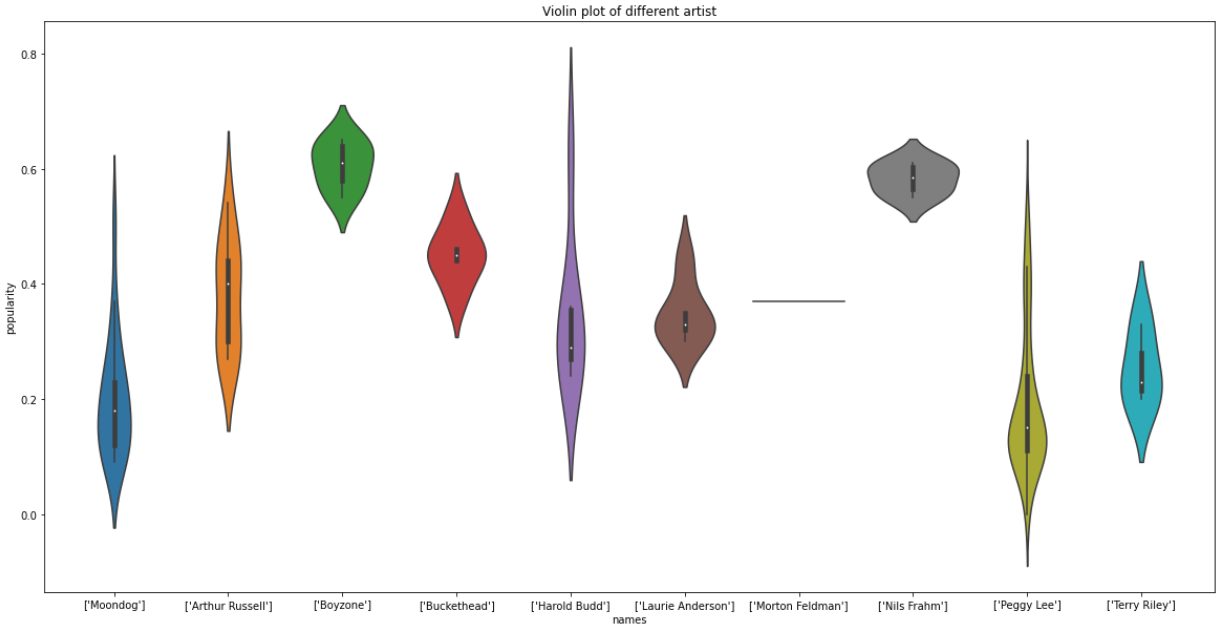


4.3 the popularity of songs composed by different artist

In [5]:
```
plt.figure(figsize=(20, 10))
ax = sns.violinplot(x='artist_names', y='popularity', data=avantgarde)
```

```
ax.set_title('Violin plot of different artist')
ax.set_xlabel('names')
ax.set_ylabel('popularity')
plt.plot()
```

Out[5]: []



5. Solution

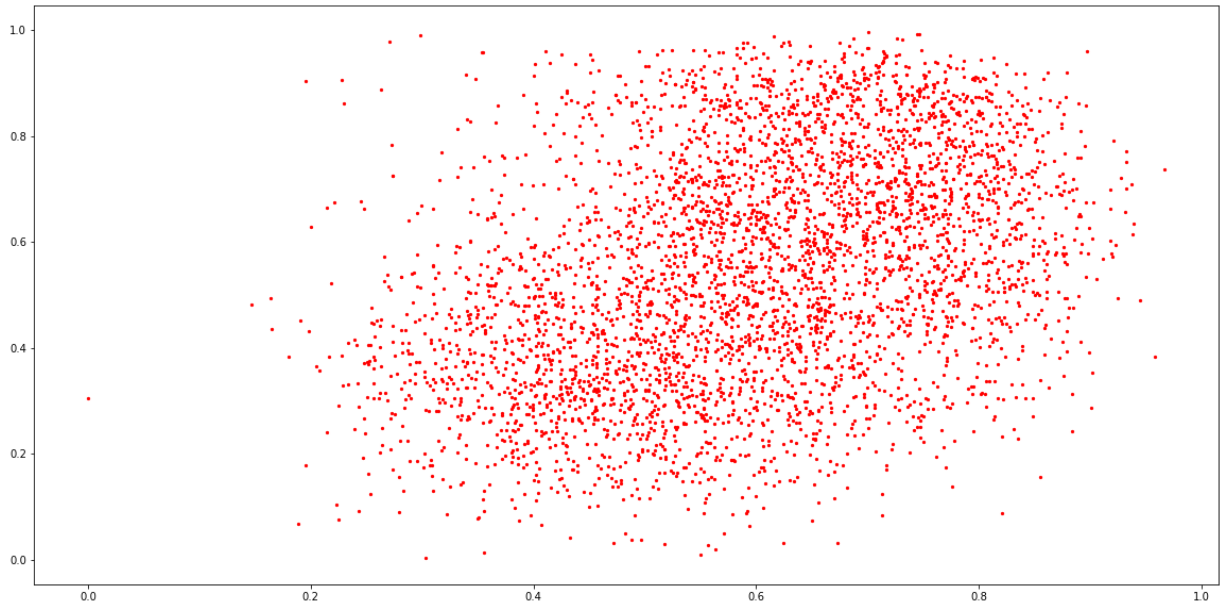
5.1 Clustering

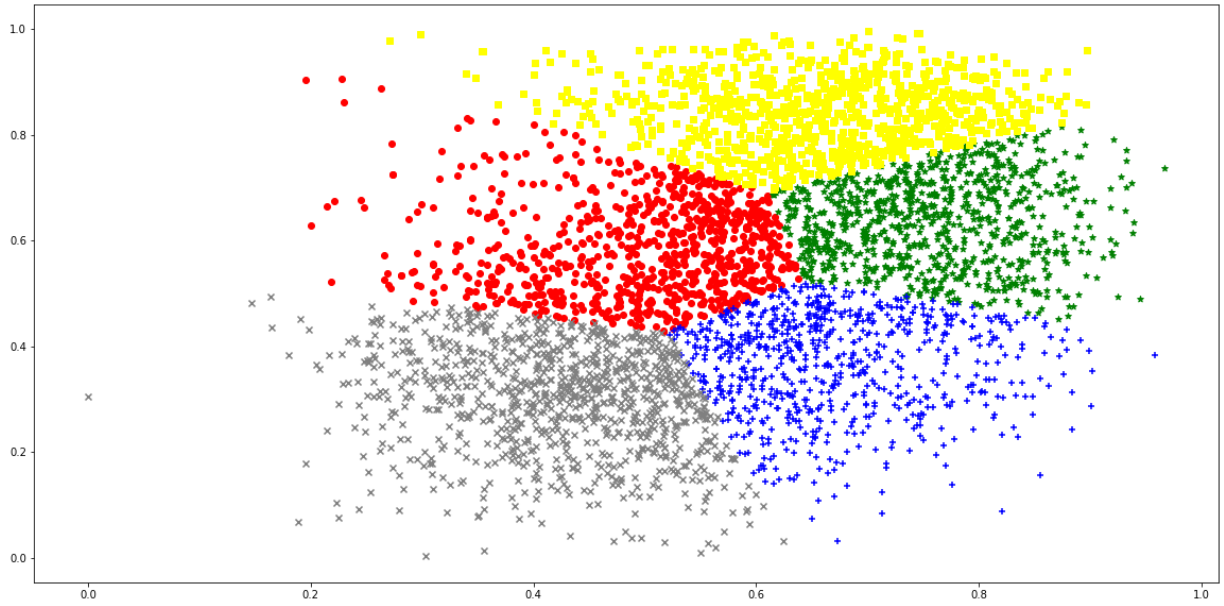
try to find out the potential genre within LATIN music（2 dimensions）

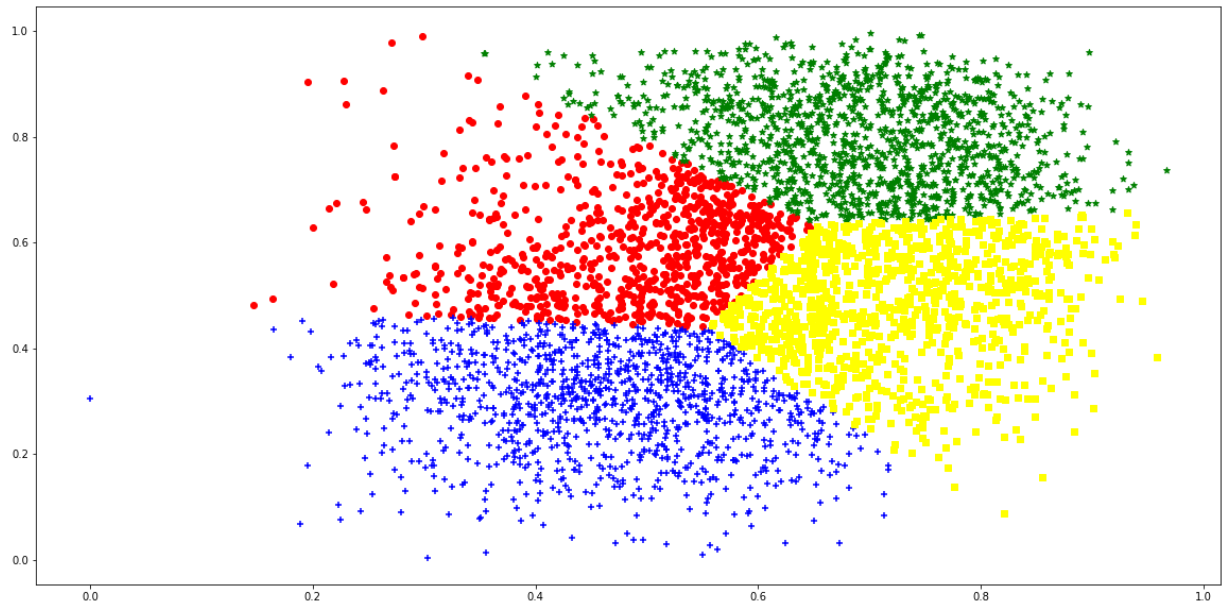
In [6]:
```
from sklearn.cluster import KMeans
x=latin.loc[:,'danceability':'energy']
```

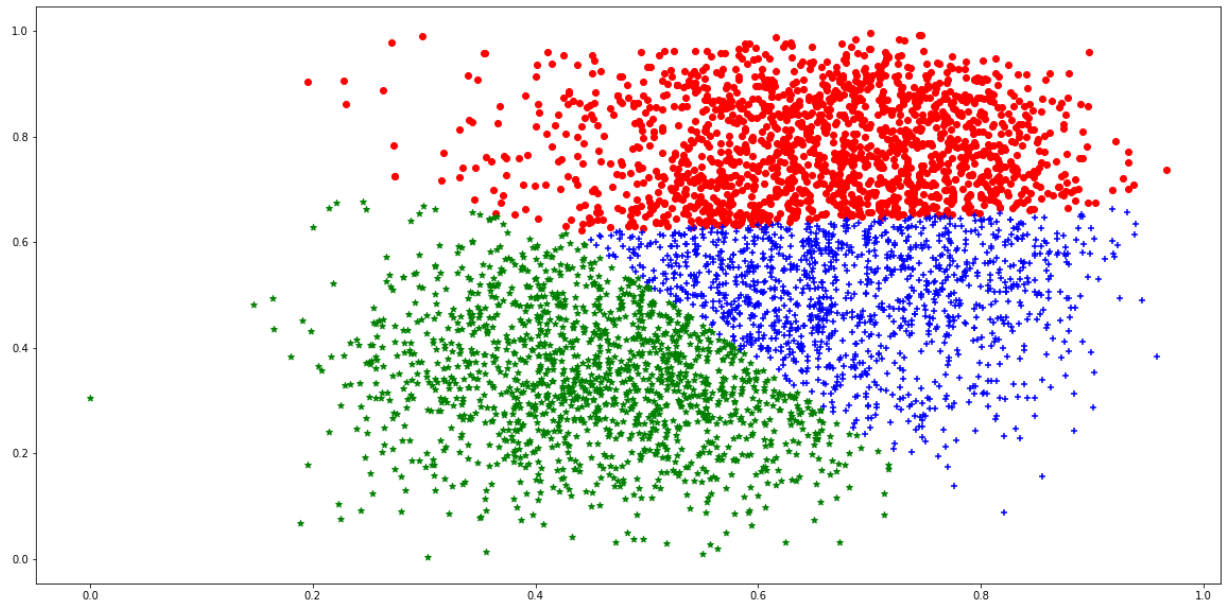
In [7]:
```
plt.figure(figsize=(20, 10))
plt.scatter(x.loc[:,'danceability'], x.loc[:,'energy'], c = "red" ,s=5,marker='o', la
plt.show()
estimator = KMeans(n_clusters=5)
estimator.fit(x)
label_pred = estimator.labels_
plt.figure(figsize=(20,10))
x0 = x[label_pred == 0]
x1 = x[label_pred == 1]
x2 = x[label_pred == 2]
x3 = x[label_pred == 3]
x4 = x[label_pred == 4]
plt.scatter(x0.loc[:, 'danceability'], x0.loc[:, 'energy'], c = "red", marker='o')
plt.scatter(x1.loc[:, 'danceability'], x1.loc[:,'energy'], c = "green", marker='*')
plt.scatter(x2.loc[:, 'danceability'], x2.loc[:, 'energy'], c = "blue", marker='+')
plt.scatter(x3.loc[:, 'danceability'], x3.loc[:, 'energy'], c = "yellow", marker=',')
plt.scatter(x4.loc[:, 'danceability'], x4.loc[:, 'energy'], c = "grey",marker='x')
plt.show()
estimator = KMeans(n_clusters=4)
estimator.fit(x)
label_pred = estimator.labels_
plt.figure(figsize=(20,10))
x0 = x[label_pred == 0]
x1 = x[label_pred == 1]
x2 = x[label_pred == 2]
x3 = x[label_pred == 3]
```

```
plt.scatter(x0.loc[:, 'danceability'], x0.loc[:, 'energy'], c = "red", marker='o')
plt.scatter(x1.loc[:, 'danceability'], x1.loc[:,'energy'], c = "green", marker='*')
plt.scatter(x2.loc[:, 'danceability'], x2.loc[:, 'energy'], c = "blue", marker='+')
plt.scatter(x3.loc[:, 'danceability'], x3.loc[:, 'energy'], c = "yellow", marker=',')
plt.show()
estimator = KMeans(n_clusters=3)
estimator.fit(x)
label_pred = estimator.labels_
plt.figure(figsize=(20,10))
x0 = x[label_pred == 0]
x1 = x[label_pred == 1]
x2 = x[label_pred == 2]
plt.scatter(x0.loc[:, 'danceability'], x0.loc[:, 'energy'], c = "red", marker='o')
plt.scatter(x1.loc[:, 'danceability'], x1.loc[:,'energy'], c = "green", marker='*')
plt.scatter(x2.loc[:, 'danceability'], x2.loc[:, 'energy'], c = "blue", marker='+')
plt.show()
```









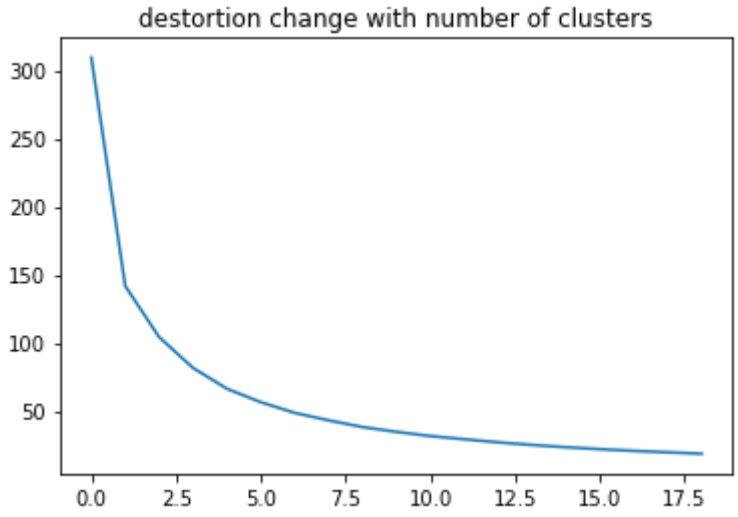
two method to estimate the effect of clustering:

- Destortion：Represents the sum of the distance from a point in the cluster to the middle of the cluster, and represents the refinement of clustering.The larger the destortion, the rougher the clustering
- Silhoutte Coefficient：Represents the measure that the distance between samples of the same class is minimized and the distance between samples of different classes is maximum, that is, whether the clustering is close and whether the distinction between classes is obvious.The larger the silhoutte coefficient, the better the clustering

In [8]:
```
#use destortion
def destortion(m,d):
    model = KMeans(n_clusters=m)
    model.fit(d)
    return model.inertia_
```

In [9]:
```
des=[]
for i in range(1,20):
    des.append(destortion(i,x))
plt.title('destortion change with number of clusters')
plt.plot(des)
```

Out[9]: [<matplotlib.lines.Line2D at 0x2a285e07dc0>]

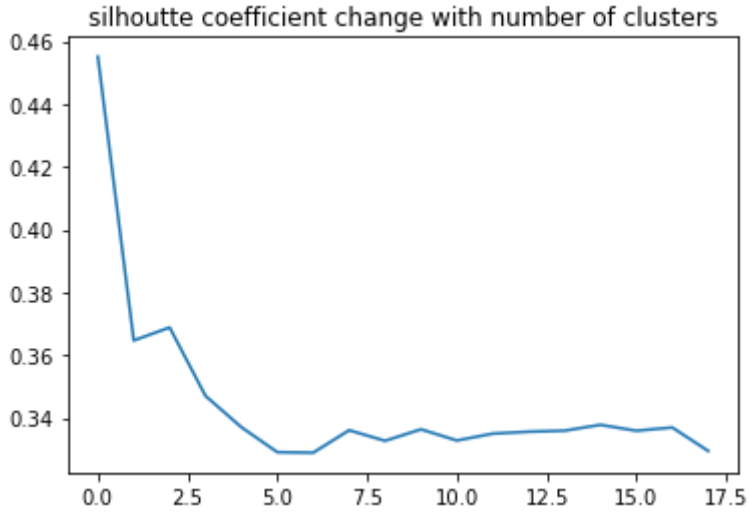


it implies that set five clusters is good enough.

In [10]:
```
#use silhouette coefficient
from sklearn.metrics import silhouette_score
def silhouette(m,d):
    model = KMeans(n_clusters=m)
    model.fit(d)
    return silhouette_score(d, model.labels_)
```

In [11]:
```
sil=[]
for i in range(2,20):
    sil.append(silhouette(i,x))
plt.title('silhoutte coefficient change with number of clusters')
plt.plot(sil)
```

Out[11]: [<matplotlib.lines.Line2D at 0x2a2863ca370>]



the silhoutte coefficient is quite low. it implies that the cluster result isn't satisfying, no matter how many clusters we choose(from 2 to 20).

5.2 regression

Observe the trend of Aerosmith's music's energy.

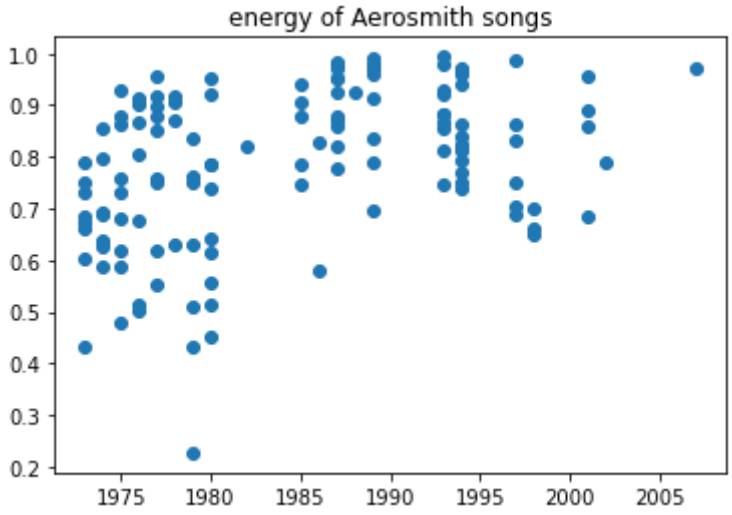
In [12]:
```
aerosmith=poprock.loc[2864:2995,]
year=aerosmith['year']
ene=aerosmith['energy']
```

```
da=pd.concat([year,ene],axis=1)
print(aerosmith)
```

```
      Unnamed: 0  Unnamed: 0.1   artist_names artists_id  danceability  \
2864        5794          5794  ['Aerosmith']   [604852]         0.425
2865        5795          5795  ['Aerosmith']   [604852]         0.386
2866        5796          5796  ['Aerosmith']   [604852]         0.516
2867        5797          5797  ['Aerosmith']   [604852]         0.409
2868        5798          5798  ['Aerosmith']   [604852]         0.469
...          ...           ...            ...        ...           ...
2991        5921          5921  ['Aerosmith']   [604852]         0.536
2992        5922          5922  ['Aerosmith']   [604852]         0.609
2993        5923          5923  ['Aerosmith']   [604852]         0.354
2994        5924          5924  ['Aerosmith']   [604852]         0.454
2995        5925          5925  ['Aerosmith']   [604852]         0.387

      energy  valence  tempo  loudness  mode  ...  instrumentalness  liveness  \
2864   0.922    0.771    0.8  0.843167     1  ...          0.019000    0.0773
2865   0.939    0.448    1.0  0.829917     1  ...          0.000007    0.3160
2866   0.971    0.805    0.8  0.923017     1  ...          0.000029    0.3700
2867   0.926    0.889    0.8  0.843767     1  ...          0.000042    0.3470
2868   0.993    0.140    0.6  0.952200     1  ...          0.001050    0.1310
...      ...      ...    ...       ...   ...  ...               ...       ...
2991   0.976    0.601    0.8  0.936583     1  ...          0.000312    0.3580
2992   0.923    0.844    0.6  0.949817     1  ...          0.000839    0.2060
2993   0.453    0.275    0.4  0.811733     1  ...          0.000048    0.3210
2994   0.688    0.322    0.8  0.909133     1  ...          0.000000    0.0677
2995   0.686    0.171    0.8  0.910533     1  ...          0.000000    0.1220

      speechiness  explicit  duration_ms  popularity  year  release_date  \
2864       0.0426         0         0.75        0.29  1980    11/11/1980
2865       0.0560         0         0.50        0.34  1985      1/1/1985
2866       0.0630         0         0.75        0.36  1987      1/1/1987
2867       0.0326         0         0.75        0.32  1988    11/15/1988
2868       0.0788         0         0.75        0.38  1989      1/1/1989
...           ...       ...          ...         ...   ...           ...
2991       0.0329         0         0.75        0.39  1989      1/1/1989
2992       0.0504         0         0.50        0.46  1993      1/1/1993
2993       0.0257         0         0.75        0.32  1980    11/11/1980
2994       0.0344         0         1.00        0.51  1997     3/18/1997
2995       0.0363         0         1.00        0.60  2001      3/5/2001

            song_title (censored)     genre
2864           Back In the Saddle  Pop/Rock
2865  Let The Music Do The Talking  Pop/Rock
2866           Permanent Vacation  Pop/Rock
2867         Chip Away The Stone   Pop/Rock
2868                  Young ****   Pop/Rock
...                          ...       ...
2991                  F.I.N.E.    Pop/Rock
2992                 Get A Grip   Pop/Rock
2993                   Dream On   Pop/Rock
2994                Full Circle   Pop/Rock
2995          Fly Away From Here   Pop/Rock

[132 rows x 22 columns]
```

In [13]:
```
plt.title('energy of Aerosmith songs')
plt.scatter(year,ene)
```

Out[13]:
```
<matplotlib.collections.PathCollection at 0x2a2863e9160>
```



In [14]:
```
import statsmodels.formula.api as smf
```

In [15]:
```
model = smf.ols('energy ~ year', data=aerosmith)
results=model.fit()
results.summary()
```

Out[15]:

OLS Regression Results

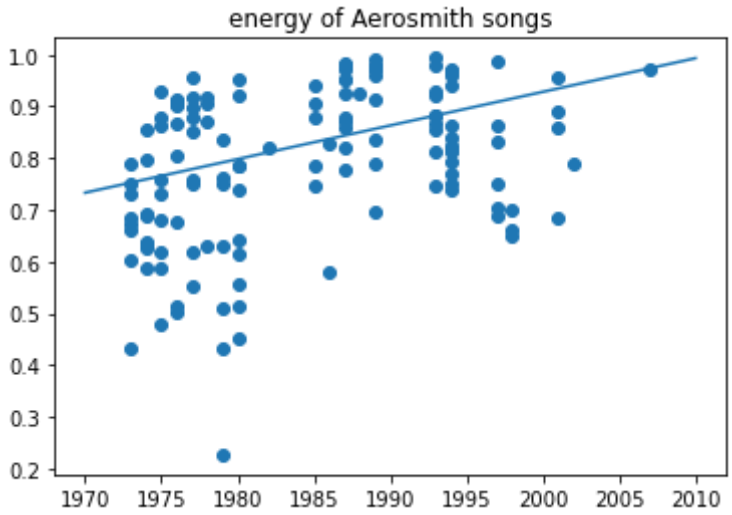
Dep. Variable:	energy	**R-squared:**	0.145
Model:	OLS	**Adj. R-squared:**	0.138
Method:	Least Squares	**F-statistic:**	21.96
Date:	Fri, 19 Feb 2021	**Prob (F-statistic):**	6.92e-06
Time:	22:41:21	**Log-Likelihood:**	74.388
No. Observations:	132	**AIC:**	-144.8
Df Residuals:	130	**BIC:**	-139.0
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-12.0719	2.745	-4.398	0.000	-17.502	-6.642
year	0.0065	0.001	4.686	0.000	0.004	0.009

Omnibus:	12.693	**Durbin-Watson:**	1.918
Prob(Omnibus):	0.002	**Jarque-Bera (JB):**	13.378
Skew:	-0.745	**Prob(JB):**	0.00124
Kurtosis:	3.461	**Cond. No.**	4.51e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.51e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [16]:
```
plt.title('energy of Aerosmith songs')
plt.scatter(year,ene)
x=np.linspace(1970,2010,5)
y=-12.0719+0.0065*x
plt.plot(x,y)
plt.show()
```



the effect is not satisfying.

use the average energy of songs of each year to represent the over-all energy of each year.

In [17]:
```
aero=pd.read_csv('D:/美赛2021/D/2021_ICM_Problem_D_Data/按流派（歌）/aerosmith.csv')
model = smf.ols('energy ~ year', data=aero)
results=model.fit()
results.summary()
```

Out[17]:

OLS Regression Results

Dep. Variable:	energy	**R-squared:**	0.784
Model:	OLS	**Adj. R-squared:**	0.773
Method:	Least Squares	**F-statistic:**	69.09
Date:	Fri, 19 Feb 2021	**Prob (F-statistic):**	9.45e-08
Time:	22:41:21	**Log-Likelihood:**	36.960
No. Observations:	21	**AIC:**	-69.92
Df Residuals:	19	**BIC:**	-67.83
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-15.1261	1.905	-7.942	0.000	-19.112	-11.140
year	0.0080	0.001	8.312	0.000	0.006	0.010

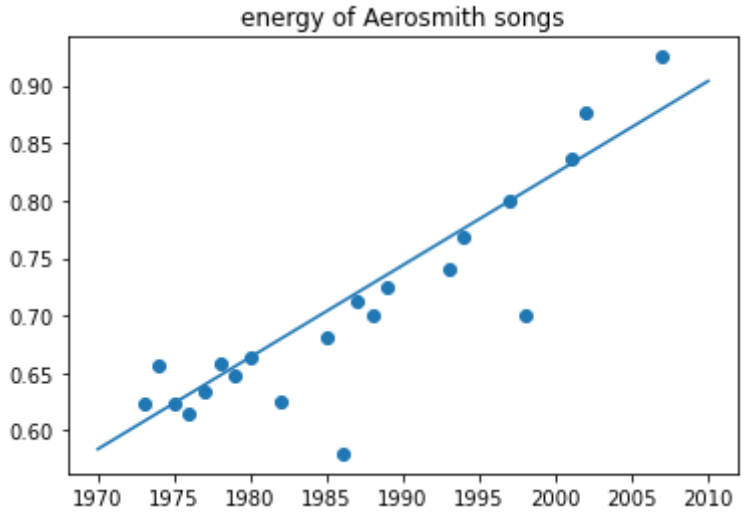
Omnibus:	11.646	**Durbin-Watson:**	1.672
Prob(Omnibus):	0.003	**Jarque-Bera (JB):**	9.463
Skew:	-1.350	**Prob(JB):**	0.00881
Kurtosis:	4.877	**Cond. No.**	3.96e+05

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.96e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [18]:
```
plt.title('energy of Aerosmith songs')
plt.scatter(aero['year'],aero['energy'])
x=np.linspace(1970,2010,5)
y=-15.1761+0.0080*x
plt.plot(x,y)
plt.show()
```



5.3 classification

attempt to train a classification method to distinguish between Latin and classical music

In [19]:
```
landc=latin.append(classical)
panda=poprock.append(avantgarde)
from sklearn import svm
from sklearn.model_selection import train_test_split
```

In [20]:
```
#pick some of the data to test
x,y = np.split(landc,indices_or_sections=(21,),axis=1)
x=x.loc[:,'danceability':'loudness']
train_data,test_data,train_label,test_label = train_test_split(x,y,random_state=1,tra
print(train_data)
train_data=np.array(train_data)
test_data=np.array(test_data)
train_label=np.array(train_label)
test_label=np.array(test_label)
```

```
     danceability  energy valence tempo  loudness
139         0.183  0.0821   0.181     1  0.648717
4090        0.639   0.358   0.426   0.6  0.855033
3589         0.71   0.675   0.953   0.8  0.874033
3999        0.438   0.764   0.445   0.6  0.858133
2282        0.257   0.421    0.53     1  0.811733
...           ...     ...     ...   ...       ...
905         0.628    0.74   0.708   0.8  0.891517
788         0.519   0.251   0.521   0.8  0.722683
3980        0.595   0.238   0.328   0.8  0.785633
235         0.335   0.299   0.382     1  0.864183
```

```
753          0.484    0.128    0.297    0.8   0.678133

[3969 rows x 5 columns]
```

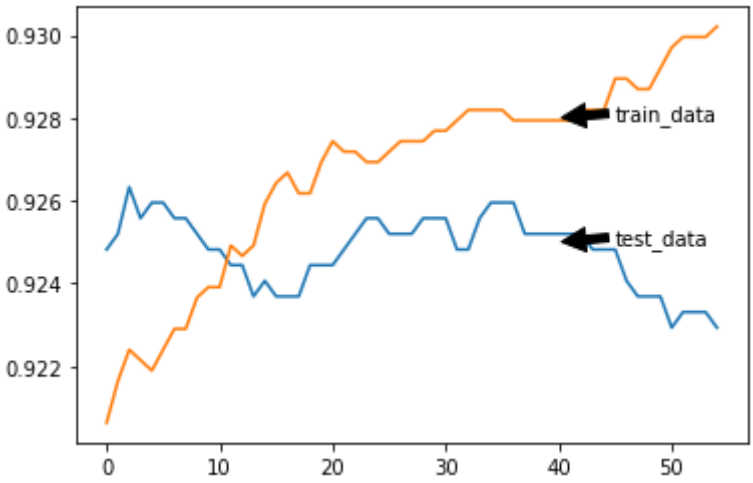
In [21]:
```
classifier = svm.SVC(C=2,kernel='rbf',gamma=10,decision_function_shape='ovo')
classifier.fit(train_data,train_label.ravel())#ravel函数在降维时默认是行序优先
print("precision on train_data: ",classifier.score(train_data,train_label))
print("precision on test_data: ",classifier.score(test_data,test_label))
print('train_decision_function:\n',classifier.decision_function(train_data))
print('predict_result:\n',classifier.predict(train_data))
```

```
precision on train_data:  0.9261778785588309
precision on test_data:  0.9244427653947865
train_decision_function:
 [-1.16312534  1.80417399  1.04132195 ...  1.6241744   1.51388194
 -0.91119934]
predict_result:
 ['Classical' 'Latin' 'Latin' ... 'Latin' 'Latin' 'Classical']
```

In [22]:
```
def test_score(m):
    classifier = svm.SVC(C=m,kernel='rbf',gamma=10,decision_function_shape='ovo')
    classifier.fit(train_data,train_label.ravel())
    return classifier.score(test_data,test_label)
def train_score(m):
    classifier = svm.SVC(C=m,kernel='rbf',gamma=10,decision_function_shape='ovo')
    classifier.fit(train_data,train_label.ravel())
    return classifier.score(train_data,train_label)
```

In [23]:
```
te_score=[]
tra_score=[]
for n in np.linspace(0.5,5,num=55):
    te_score.append(test_score(n))
    tra_score.append(train_score(n))
```

In [24]:
```
plt.plot(te_score)
plt.plot(tra_score)
plt.annotate('train_data', xy=(40, 0.928), xytext=(45, 0.928), arrowprops=dict(faceco
plt.annotate('test_data', xy=(40, 0.925), xytext=(45, 0.925), arrowprops=dict(facecol
plt.show()
```



According to the figure above, the penalty factor is set between 1.5 and 4 to have similar effect.When the penalty coefficient is less than 1.5, there is a certain degree of underfitting, while when the penalty coefficient is close to 5, there is a more serious overfitting, and the accuracy rate of the training set increases significantly while the accuracy rate of the test set decreases significantly.

6. Conclusion

- In order to analyze the potential hidden rules in the data set, I simply used the methods of clustering, regression and classification to analyze the specific data, and observed the overall rules of part of the data through data visualization, and reached the following conclusions.

- Visualization:

- In terms of danceability, Latin performed the best, followed by Pop/Rock and Classical performed the worst.This is consistent with what we think about music types. ** In terms of data distribution, Latin's danceability and energy are concentrated in the range of 0.6-0.8.On the one hand, Latin is relatively lively, and on the other hand, it indicates that there may be a certain correlation between danceability and energy
- In terms of the popularity of songs written by artists, it's easy to see which artists have always produced very popular music, and which artists have always been less well-known.Others fluctuate, perhaps because the well-known artist has accidentally created work that is at odds with the mainstream aesthetic.
- Cluster:
- Latin music in a dataset can potentially be divided into five different categories
- However, from the perspective of Silhoutte coefficient, no matter what the clustering method is, it is not very satisfactory.This may be due to a defect in the data set itself.

- Regression:

- Aerosmith's music has a tendency to fill up with energy over time.We can predict that if they release a song in 2021, the odds are that it will be traditional, intense hard rock, and maybe even metal. LOL

- Classification:

- SVM method can be used to find a more reasonable classification method to distinguish Latin and classical music, and the penalty coefficient is best set between 1.5 and 4

7. Assessment

- Music is notoriously difficult to measure with a few simple data sets, especially since the data in this dataset basically describe the general characteristics of songs with few details.Such a rough description is likely to lead to a false understanding of the songs, or overestimate the similarities between songs and underestimate the differences between songs.This is also an important reason for the unsatisfactory clustering effect.
- Due to the large amount of data in songs, due to the limited computing power, it is temporarily impossible to conduct sufficient research on all types of songs.
- More than 10 dimensions of songs are described in the dataset, but it is difficult to deal with such high-dimensional data, whether for clustering, regression or classification, especially for visualization above 3 dimensions.This is particularly deadly for clustering.In the actual operation, I chose two dimensions to practice
- There may be a strong correlation between attributes, and once the PCA method is used, the newly acquired dimension will lose its realistic significance.