```
min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_train_minmax
```

```
scaler = preprocessing.StandardScaler().fit(X)

scaler.mean_
array([ 1. ...,  0. ...,  0.33...])

scaler.std_
```

```
scaler.transform(X)

#可以直接使用训练集对测试集数据进行转换
scaler.transform([[-1.,  1., 0.]])
```

kmeans

```
# determine the best number of clusters
clusters = range(2, 30)
inertias = []
silhouettes = []

for n_clusters in clusters:
    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', random_state=42, n_jobs=-1)
    kmeans = kmeans.fit(feature)
    label = kmeans.predict(feature)

    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(feature, label))
```

choose best K (i.e., number of clusters)

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score

inertias = []
silhouettes = []

ks = range(2,30)
for k in ks:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(items_rotated)

    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(items_rotated, kmeans.predict(items_rotated)))


# visualization
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(18, 6))
ax[0].plot(clusters, inertias, 'o-', label='Sum of Squared Distances')
ax[0].grid(True)
ax[1].plot(clusters, silhouettes, 'o-', label='Silhouette Coefficient')
ax[1].grid(True)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```

similarity matrix

```
from sklearn.preprocessing import normalize

# Step 1: build the Song-User matrix
song_user = data.groupby(['song_played', 'user_id'])['id'].count().unstack(fill_value=0)
song_user = (song_user > 0).astype(int)

song_user.head()

# Step 2: build song-song similarity matrix
song_user_norm = normalize(song_user, axis=1)  # normalize the song-user matrix
similarity = np.dot(song_user_norm, song_user_norm.T)  # calculate the similarity matrix
similarity_df = pd.DataFrame(similarity, index=song_user.index, columns=song_user.index)

similarity_df.head()
```