Data 144 Final Project: Spotify Song Recommender

Davis Ulrich

```
import pandas as pd
import collections
from sklearn.cluster import KMeans
from sklearn.cluster import SpectralClustering
from sklearn.metrics import silhouette_score
import numpy as np
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA

import matplotlib
import matplotlib.pyplot as plt
```

Main DataFrame

```
data_main = pd.read_csv('data.csv')
```

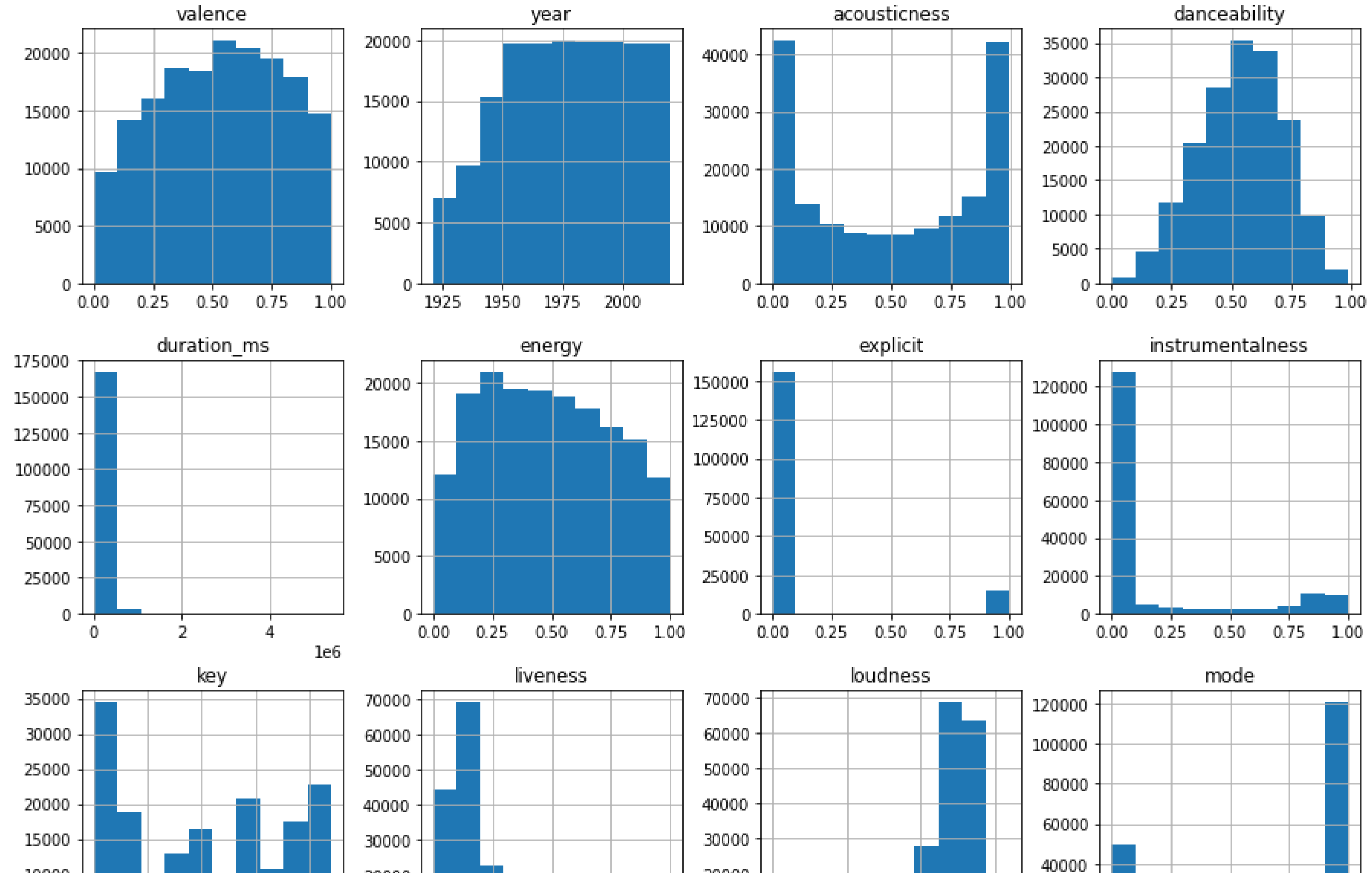
```
# Main data frame and its shape

print(data_main.shape)
data_main.head()
```

```
(170653, 19)
```

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrx
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNty

```
data_main.hist(figsize=(15, 15));
```




```
# The release date is redundant and not in a usable type, the id is useless if we have a number instead

data_main = data_main.drop(columns = ['release_date', 'id'])
```

popularity speechiness tempo

```
# Replace song id string with a number and create a dictionary mapping the song number to the name (so we ca

data_main = data_main.reset_index()
song_dict = dict(zip(data_main['index'], data_main['name']))
for i in range(3):
    print(i, ': ', song_dict[i])

# data_main = data_main.drop(columns = ['name'])
data_main.head()
```

```
0 :  Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve
1 :  Clancy Lowered the Boom
2 :  Gati Bali
```

	index	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	instrum
0	0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	
1	1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	

```
# Lets replace the artist feature with a unique value for each artist (or list of artists)

unique_artists = data_main['artists'].unique()
artist_dict = dict(zip(range(len(unique_artists)), unique_artists))

for i in range(111, 114):
```

```
print(i, ': ', artist_dict[i])
```

```
111 :  ['George Olsen']
112 :  ['Jailess']
113 :  ['Duke Ellington & His Washingtonians']
```

```
# We need a new dict to set up the artist_id column

dict_for_artists_column = dict(zip(unique_artists, range(len(unique_artists))))

data_main['artist_id'] = [dict_for_artists_column[artist] for artist in data_main['artists']]
data_main = data_main.rename(columns = {'index': 'song_id'})

data_main.head()
```

	song_id	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	instr
0	0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	
1	1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	

Model Designing:

Model DF:

```
# The dataframe above seems to be too big for tsne to handle so I'm gonna reduce the number of rows based or
# Used for official model: model_df_with_names = data_main[data_main['popularity'] > 40].sample(9000, random

model_df_with_names = data_main[data_main['popularity'] > 40].sample(9000, random_state=42)
model_df_unnormalized = model_df_with_names.drop(columns = ['artists', 'song_id', 'name'])

model_df = model_df_unnormalized
model_df['duration_ms'] = (model_df['duration_ms'] - model_df['duration_ms'].mean()) / model_df['duration_ms
# model_df = (model_df_unnormalized - model_df_unnormalized.mean()) / model_df_unnormalized.std()

model_df
```

	valence	year	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key
90583	0.4250	2012	0.69300	0.615	-0.730106	0.557	0	0.000000	8
38582	0.1390	2020	0.04350	0.764	-0.383929	0.502	0	0.000000	5

PCA:

```
pca_reduced = PCA(n_components = 5, random_state=42).fit_transform(model_df.values)
pca_reduced
```

```
array([[ 9.64621800e+02,  2.33811113e-01, -1.37523865e+01,
         2.87389007e+00,  1.31953609e+00],
       [-2.53337333e+03, -4.00582544e+01, -1.53987067e+01,
        -1.66271655e+01,  9.08861345e-01],
       [-3.47237656e+03, -4.84614845e+00, -1.53346651e+01,
        -4.31331885e+00, -1.79102038e-02],
       ...,
       [ 1.35056029e+04, -4.95282249e+01,  1.98721131e+01,
        -1.70294941e+00, -2.36548375e-02],
       [-6.67140337e+03, -1.76852401e+01,  1.69861584e+01,
         9.87897793e+00, -1.35689116e+00],
       [-3.76537792e+03, -6.98443758e+00, -1.49567965e+01,
        -8.78641697e-01, -1.26577697e+00]])
```

Spectral Clustering model before t-SNE:

```
# The spectral clustering model

# spectral = SpectralClustering(n_clusters= 10, random_state = 42).fit(pca_reduced)


# Cluster labels

# spectral.labels_[:10]


# plt.hist(spectral.labels_)
```

Choosing the best k:

```
# Taken from https://github.com/ciortanmadalina/high_noise_clustering/blob/master/spectral_clustering.ipynb

import scipy
from scipy.sparse import csgraph
# from scipy.sparse.linalg import eigsh
from numpy import linalg as LA
def eigenDecomposition(A, plot = True, topK = 5):
    """
    :param A: Affinity matrix
    :param plot: plots the sorted eigen values for visual inspection
    :return A tuple containing:
    - the optimal number of clusters by eigengap heuristic
    - all eigen values
    - all eigen vectors

    This method performs the eigen decomposition on a given affinity matrix,
    following the steps recommended in the paper:
```

```
    following the steps recommended in the paper:
    1. Construct the normalized affinity matrix: L = D−1/2AD^ −1/2.
    2. Find the eigenvalues and their associated eigen vectors
    3. Identify the maximum gap which corresponds to the number of clusters
    by eigengap heuristic

    References:
    https://papers.nips.cc/paper/2619-self-tuning-spectral-clustering.pdf
    http://www.kyb.mpg.de/fileadmin/user_upload/files/publications/attachments/Luxburg07_tutorial_4488%5b0%5
    """
    L = csgraph.laplacian(A, normed=True)
    n_components = A.shape[0]

    # LM parameter : Eigenvalues with largest magnitude (eigs, eigsh), that is, largest eigenvalues in
    # the euclidean norm of complex numbers.
#     eigenvalues, eigenvectors = eigsh(L, k=n_components, which="LM", sigma=1.0, maxiter=5000)
    eigenvalues, eigenvectors = LA.eig(L)

    if plot:
        plt.title('Largest eigen values of input matrix')
        plt.scatter(np.arange(len(eigenvalues)), eigenvalues)
        plt.grid()

    # Identify the optimal number of clusters as the index corresponding
    # to the larger gap between eigen values
    index_largest_gap = np.argsort(np.diff(eigenvalues))[::-1][:topK]
    nb_clusters = index_largest_gap + 1

    return nb_clusters, eigenvalues, eigenvectors


# optimal_k, _, _ = eigenDecomposition(spectral.affinity_matrix_)
# optimal_k


# optimal_k, _, _ = eigenDecomposition(spectral_after_tsne.affinity_matrix_)
# optimal_k
```

Dimentionality Reduction: t-SNE

```
#dim_reduc = TSNE(n_components=2, perplexity=30, verbose=2, method='barnes_hut', n_iter = 500, random_state=
dim_reduc = TSNE(n_components=2, perplexity=30, verbose=2, method='barnes_hut', n_iter = 500, random_state=4

## Parameters
## n_components = number of dimensions you want your data to be reduced
## perplexity =  Number of neighboours to fit the gaussian , normally 30
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 9000 samples in 0.016s...
[t-SNE] Computed neighbors for 9000 samples in 0.157s...
[t-SNE] Computed conditional probabilities for sample 1000 / 9000
[t-SNE] Computed conditional probabilities for sample 2000 / 9000
[t-SNE] Computed conditional probabilities for sample 3000 / 9000
[t-SNE] Computed conditional probabilities for sample 4000 / 9000
[t-SNE] Computed conditional probabilities for sample 5000 / 9000
[t-SNE] Computed conditional probabilities for sample 6000 / 9000
[t-SNE] Computed conditional probabilities for sample 7000 / 9000
[t-SNE] Computed conditional probabilities for sample 8000 / 9000
[t-SNE] Computed conditional probabilities for sample 9000 / 9000
```

```
[t-SNE] Mean sigma: 10.447924
[t-SNE] Computed conditional probabilities in 0.665s
[t-SNE] Iteration 50: error = 88.4570923, gradient norm = 0.0276710 (50 iterations in 4.668s)
[t-SNE] Iteration 100: error = 71.9435272, gradient norm = 0.0071518 (50 iterations in 3.294s)
[t-SNE] Iteration 150: error = 66.8732300, gradient norm = 0.0055065 (50 iterations in 3.166s)
[t-SNE] Iteration 200: error = 64.0362701, gradient norm = 0.0038939 (50 iterations in 3.098s)
[t-SNE] Iteration 250: error = 62.1404572, gradient norm = 0.0033221 (50 iterations in 3.040s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 62.140457
[t-SNE] Iteration 300: error = 1.9775333, gradient norm = 0.0013072 (50 iterations in 3.145s)
[t-SNE] Iteration 350: error = 1.3936312, gradient norm = 0.0006324 (50 iterations in 3.293s)
[t-SNE] Iteration 400: error = 1.0951393, gradient norm = 0.0003801 (50 iterations in 3.341s)
[t-SNE] Iteration 450: error = 0.9239067, gradient norm = 0.0002617 (50 iterations in 3.326s)
[t-SNE] Iteration 500: error = 0.8146366, gradient norm = 0.0001945 (50 iterations in 3.293s)
[t-SNE] KL divergence after 500 iterations: 0.814637
```

```
# Creating the dataframe to use in the more advanced visual tool

plot_tool_df = pd.DataFrame(dim_reduc).rename({0: 'x', 1: 'y'}, axis = 1)
# plot_tool_df['cluster'] = spectral.labels_
plot_tool_df
```

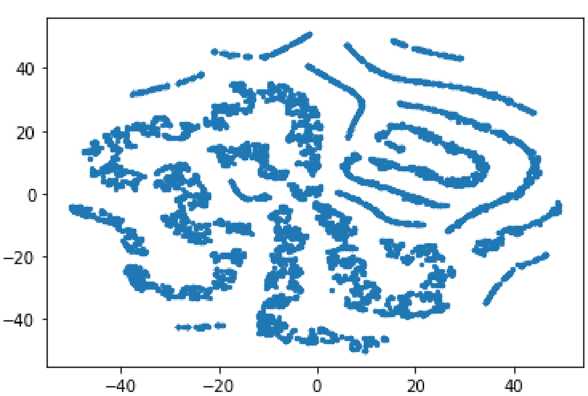
	x	y
0	29.475552	13.745345
1	-48.286602	-4.050319
2	-19.697844	10.872694
3	-3.811779	48.944164
4	34.335938	8.841171
...	...	...
8995	-27.986593	8.401427
8996	-15.053980	15.359935
8997	6.647834	19.993357
8998	9.850985	-48.479439
8999	-1.939630	0.670322

9000 rows × 2 columns

```
# random state 42

x_axis= dim_reduc[:,0]
y_axis= dim_reduc[:,1]

plt.scatter(x_axis, y_axis, s=5)
plt.show() ## The plots vary each time you run them
```



Spectral after t-SNE:

```
spectral_after_tsne = SpectralClustering(n_clusters=22, random_state=42).fit(dim_reduc)
```

```
spectral_after_tsne.labels_
```

```
array([ 1, 11,  4, ..., 18,  9, 16], dtype=int32)
```

```
plot_tool_df['cluster'] = spectral_after_tsne.labels_
plot_tool_df['artist'] = np.array(model_df_with_names['artists'])
plot_tool_df['song'] = np.array(model_df_with_names['name'])
plot_tool_df
```

	x	y	cluster	artist	song
0	29.475552	13.745345	1	['Gyptian']	Wine Slow
1	-48.286602	-4.050319	11	['Future', 'Lil Uzi Vert']	Plastic
2	-19.697844	10.872694	4	['DJ Khaled', 'Drake', 'Rick Ross', 'Lil Wayne']	No New Friends - SFTB Remix
3	-3.811779	48.944164	21	['Dave Koz', 'Chris Botti']	Love Is On The Way
4	34.335938	8.841171	1	['NEEDTOBREATHE']	HAPPINESS
...	...	...	...	...	...
8995	-27.986593	8.401427	8	['Joan Sebastian']	El Charro Viejo
8996	-15.053980	15.359935	4	['Hans Zimmer']	Homeland
8997	6.647834	19.993357	18	['Big Mountain', 'Dave Way']	Lean on Me - Party Version
8998	9.850985	-48.479439	9	['Bob Dylan']	One More Cup of Coffee
8999	-1.939630	0.670322	16	['Lady Gaga']	You And I

9000 rows × 5 columns

```
# Save the plot_tool_df just in case the PCA comes up with diff results
```

```
# plot_tool_df.to_csv('spectral_model_1.csv', index=False)
```

d3 Scatterplot tool:

```
# Create a tab separated file for visualization purposes
```

```
plot_tool_df.to_csv('dim_reduce.tsv', sep='\t', index=False)
```

```
from google.colab.output import eval_js
from IPython.display import Javascript

!git clone https://github.com/CAHLR/d3-scatterplot.git
```

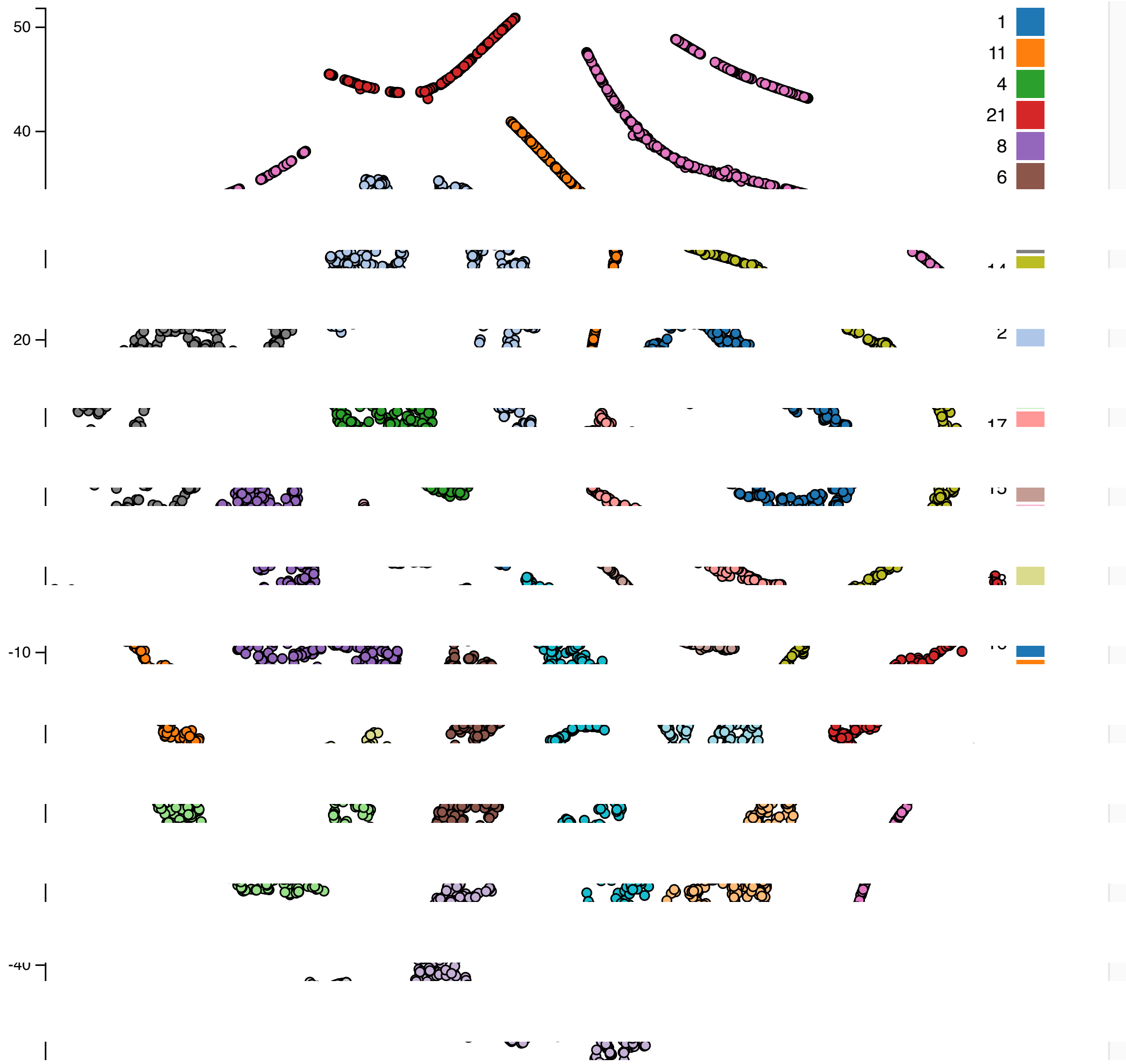
```
Cloning into 'd3-scatterplot'...
remote: Enumerating objects: 1022, done.
remote: Total 1022 (delta 0), reused 0 (delta 0), pack-reused 1022
Receiving objects: 100% (1022/1022), 1.91 MiB | 1.79 MiB/s, done.
Resolving deltas: 100% (593/593), done.
```

```
def show_port(port, data_file, width=600, height=800):
  display(Javascript("""
  (async ()=>{
    fm = document.createElement('iframe')
    fm.src = await google.colab.kernel.proxyPort(%d) + '/index.html?dataset=%s'
    fm.width = '90%%'
    fm.height = '%d'
    fm.frameBorder = 0
    document.body.append(fm)
  })();
  """ % (port, data_file, height)))

port = 8000
height = 1600


data_file = 'dim_reduce.tsv'

get_ipython().system_raw('cd d3-scatterplot && python3 -m http.server %d &' % port)
show_port(port, data_file, height)
```


50
40
20
-10
1
11
4
21
8
6
2