National Research University Higher School of Economics

Analysis of Top Spotify Tracks of 2017

Course Project, 2018/19

The team:
Anton Kukulianski, Uladzislau Bohdan
MS program “Data Science”, 1st year
Faculty of Computer Science

Moscow, 2018

```
In [1]: import pandas as pd
        import numpy as np
        from matplotlib import pyplot as plt
        from sklearn.cluster import KMeans
        import seaborn as sns
        from IPython.display import display
        %matplotlib inline
```

1 An explanation of dataset choice

Spotify is one of the largest music streaming services with 180 million monthly active users, as of July 2018. By providing services to such a huge number of people worldwide, Spotify is gathering an enormous amount of data about the playbacks, which is an amazing source of a research.

In order to complete the project we have chosen a dataset of the top listened tracks on Spotify in 2017, the dataset is available on https://www.kaggle.com/nadintamer/top-tracks-of-2017. The dataset contains 100 top listened tracks and besides tracks' names and artists contains multiple audio features such as: energy, loudness, instrumentalness, liveness, tempo and other - 13 numeric features in total, which are explained in more detail in the link with the dataset. These features are generated by Spotify algorithms. Additional data or dataset extension may be obtained using Spotify API. This data may be used to research the tracks and to attempt to explain their popularity, as well as to predict a set of properties of the top tracks in 2018.

```
In [2]: df = pd.read_csv('top-tracks-of-2017.zip')
```

2 Homework 2

Let's select 6 columns to observe. All of them are numerical and they look promising for clusterization, since we may assume some clusters even before the clustering.

```
In [3]: columns = np.array(['energy', 'loudness', 'acousticness', 'liveness',
                           'instrumentalness', 'valence'])
```

```
In [4]: truncated = df[columns].values
```

Here is help-function for clustering. We choose the best one among 10 random initializations.

```
In [5]: np.random.seed(123) # for reproductibility

        def get_best_clustering(X, n_class=5):
            inerts = []
            predictions = []
            for _ in range(10):
                km = KMeans(n_class, init='random', random_state=np.random.randint(0, 1023))
                prediction = km.fit_predict(X)
                inerts.append(km.inertia_)
                predictions.append(prediction)
            return predictions[np.argmin(inerts)]
```

```
In [6]: prediction_5 = get_best_clustering(truncated)
        prediction_9 = get_best_clustering(truncated, 9)
```

Here is a function for bulding a relative differences of means with respect to global mean.

```
In [7]: def clusters_relative(data, prediction, n_class):
            cm = sns.light_palette('blue', as_cmap=True)
            _data = np.array(
                [
                    100 * (data[prediction == klass].mean(0) - data.mean(0)) / data.mean(0)
                    for klass in range(n_class)
                ]
            )
            df = pd.DataFrame(_data, index=[f'class #{i}' for i in range(n_class)],
                              columns=columns)
            print(df.to_latex())
            return _data
```

We'll be looking for differnces grater than 25%

```
In [8]: def describe_the_data(data, n_class, border=25):
            for i in range(n_class):
                f_plus = columns[data[i] > border]
                f_minus = columns[data[i] < -border]
                print(f'Cluster #{i} description:')
                print(f"\t{', '.join(f_plus) or 'no features'}"
                      "{'are' if len(f_plus) != 1 else 'is'} much greater than the average")
                print(f"\t{', '.join(f_minus) or 'no features'}"
                      "{'are' if len(f_minus) != 1 else 'is'} much smaller than the average")
```

2.1 5-clusters clustering description

```
In [9]: data = clusters_relative(truncated, prediction_5, 5)
```

Relative to global

	energy	loudness	acousticness	liveness	instrumentalness	valence
class #0	-6.163299	11.021138	-10.963970	-5.054918	-94.256858	-9.570144
class #1	20.134145	-43.436138	3.016768	3.158741	-99.876731	24.014220
class #2	9.831941	-16.690115	-26.985671	4.998529	99.190520	11.208029
class #3	-29.952020	84.953960	26.272883	-20.481784	281.056099	-21.051970
class #4	-26.591896	39.742985	107.880151	3.641205	-51.507201	-29.983777

```
In [10]: describe_the_data(data, 5)
```

```
Cluster #0 description:
        no features are much greater than the average
        instrumentalness is much smaller than the average
Cluster #1 description:
```

```
        no features are much greater than the average
        loudness, instrumentalness are much smaller than the average
Cluster #2 description:
        instrumentalness is much greater than the average
        acousticness is much smaller than the average
Cluster #3 description:
        loudness, acousticness, instrumentalness are much greater than the average
        energy is much smaller than the average
Cluster #4 description:
        loudness, acousticness are much greater than the average
        energy, instrumentalness, valence are much smaller than the average
```

Colloquial descriptions of the clusters:

- #0: this cluster is likely to contain much rap music in its traditional form - without expressing too much positiveness or negativeness and not having rich instrumental part.
- #1: the tracks may be described as cheerful but calm at the same time
- #2: in opposite to other clusters this one is unlikely to contain much rap or hip-hop music which is generally described as lacking an instrumental part but being speechful
- #3: the cluster contains tracks with rich instrumental part and some voice parts; voice parts are not as extensive as in the rap samples; the tracks in this cluster may also be described as pretty calm
- #4: the tracks with high confidence of being acoustic; at the same time not very much cheerful or happy; may be described as "melancholic"

2.2 9-clusters clustering description

```
In [11]: data = clusters_relative(truncated, prediction_9, 9)
```

Relative to global

	energy	loudness	acousticness	liveness	instrumentalness	valence
class #0	9.582406	-13.569299	-46.509529	2.144987	358.911335	-3.696265
class #1	-35.875120	96.014259	57.340021	-25.678532	534.342896	-17.673825
class #2	-4.839745	12.157256	-30.691649	-13.848626	-99.759271	-28.319864
class #3	4.067339	-1.220003	-8.327014	21.653542	-87.768560	-5.146804
class #4	-21.402516	37.149327	179.303592	-1.645626	-89.090790	-27.942723
class #5	-16.950461	21.065341	-2.165033	-12.567145	-99.918205	11.246710
class #6	19.863034	-44.340605	7.472100	1.542936	-99.867249	21.919705
class #7	-31.162875	57.204143	-57.500155	11.561880	-8.142712	-41.699916
class #8	10.868940	-25.119192	5.746024	-3.158552	-98.929870	37.354003

```
In [12]: describe_the_data(data, 9)
```

```
Cluster #0 description:
        instrumentalness is much greater than the average
        acousticness is much smaller than the average
Cluster #1 description:
        loudness, acousticness, instrumentalness are much greater than the average
        energy, liveness are much smaller than the average
Cluster #2 description:
        no features are much greater than the average
        acousticness, instrumentalness, valence are much smaller than the average
Cluster #3 description:
        no features are much greater than the average
        instrumentalness is much smaller than the average
Cluster #4 description:
        loudness, acousticness are much greater than the average
        instrumentalness, valence are much smaller than the average
Cluster #5 description:
        no features are much greater than the average
        instrumentalness is much smaller than the average
Cluster #6 description:
        no features are much greater than the average
        loudness, instrumentalness are much smaller than the average
Cluster #7 description:
        loudness is much greater than the average
        energy, acousticness, valence are much smaller than the average
Cluster #8 description:
        valence is much greater than the average
        loudness, instrumentalness are much smaller than the average
```

Colloquial descriptions of the clusters:

- #5: a general-purpose cluster for tracks being rather speechful than instrumental
- #3: a cluster containing tracks with are characterized with presence of live audience in the recording: the recording was probably made during a live concert
- #4: calm and less prominent tracks may belong to this cluster; that does not necessarily mean tracks cannot be described as "loud"
- #0: similar to cluster #2 from 5-clusterization, these tracks are non-speechful

Other clusters... overall, due to the size of the dataset, description of each cluster in 9-clusterization cannot be meaningful enough which makes us stop describing clusters at this point.

So, obviously, clustering into 5 clusters gave us more meaningful and interesting division.

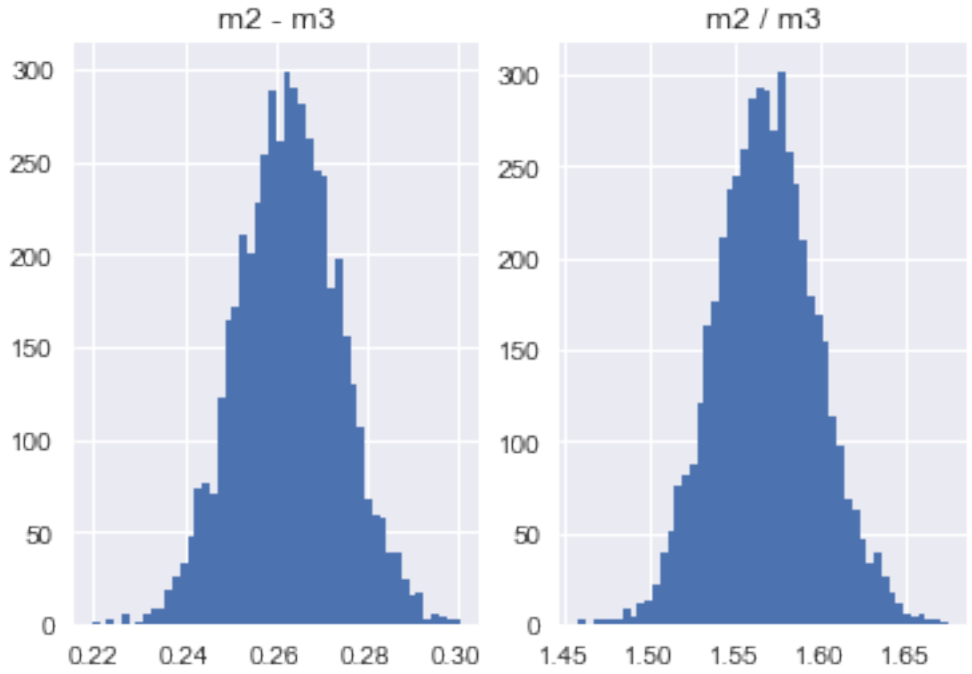
2.3 Bootstrap

```
In [13]: def intervals(x):
             return [
                 (x.mean() - 1.96 * x.std(ddof=1), x.mean() + 1.96 * x.std(ddof=1)),
```

```
            (np.percentile(x, 2.5), np.percentile(x, 97.5))
        ]
```

Let's consider energy of 2 and 3 clusters.

In [14]:
```
c2 = np.random.choice(truncated[:, 0][prediction_5==2], (5000, 100)).mean(1)
c3 = np.random.choice(truncated[:, 0][prediction_5==3], (5000, 100)).mean(1)
d = c2 - c3
q = c2 / c3

fig, ax = plt.subplots(1, 2)
ax[0].hist(d, bins=50);
ax[0].set_title('m2 - m3');
ax[1].hist(q, bins=50);
ax[1].set_title('m2 / m3');
```



Confidence intervals for differnce

In [15]:
```
dpivotal, dnonpivotal = intervals(d)

print(f'pivotal interval for d = {dpivotal}')
print(f'non-pivotal interval for d = {dnonpivotal}')
```

```
pivotal interval for d = (0.2409886467920023, 0.2850309092079979)
non-pivotal interval for d = (0.24123824999999993, 0.2853507500000001)
```

Confidence intervals for quotients

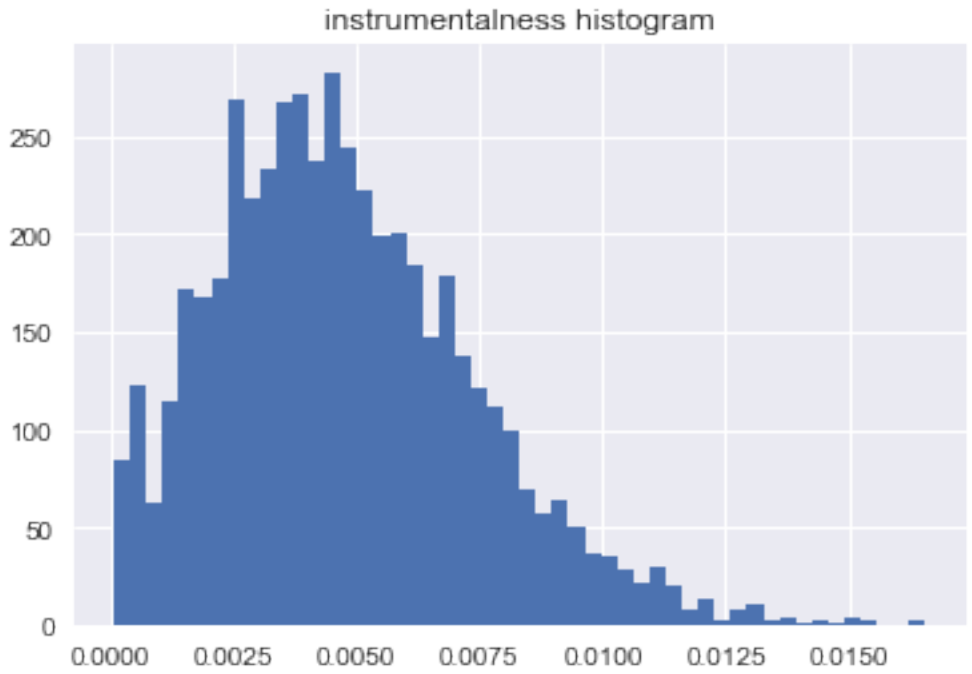
```
In [16]: qpivotal, qnonpivotal = intervals(q)

         print(f'pivotal interval for q = {qpivotal}')
         print(f'non-pivotal interval for q = {qnonpivotal}')
```

```
pivotal interval for q = (1.510685004384165, 1.6268270419871893)
non-pivotal interval for q = (1.5130346377330284, 1.6294618808593402)
```

We may notice that m_2 is always greater than m_3

Let's build confidence intervals for instrumentalness

```
In [17]: x = np.random.choice(truncated[:, 4], (5000, 100)).mean(1)
         plt.hist(x, bins=50);
         plt.title('instrumentalness histogram');
```



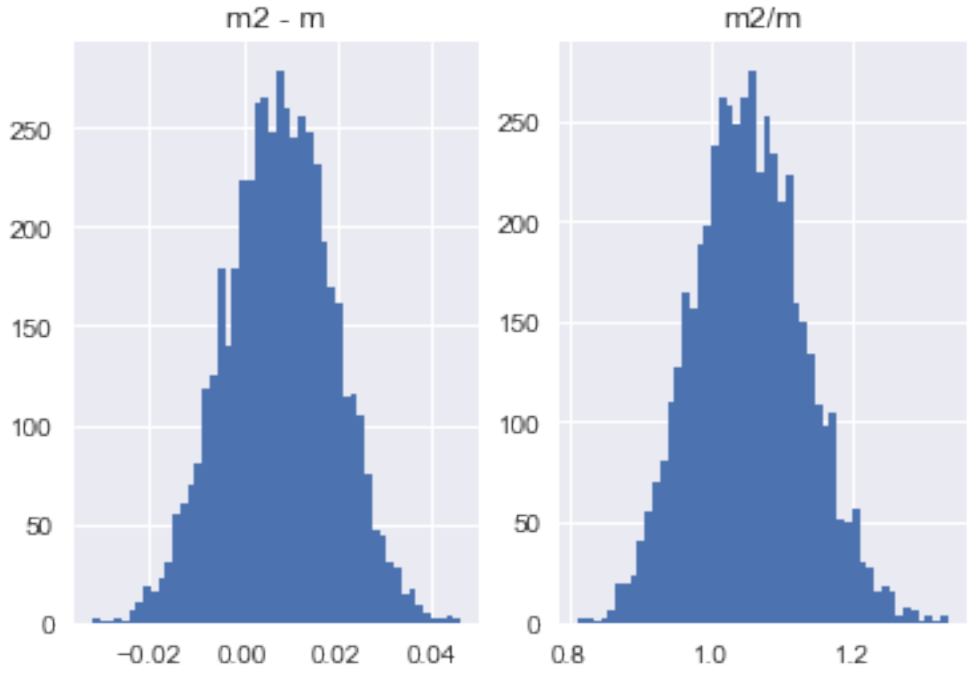
And pivotal and non-pivotal confidence intervals

```
In [18]: xpivotal, xnonpivotal = intervals(x)

         print(f'pivotal interval for instrumentalness = {xpivotal}')
         print(f'non-pivotal interval for instrumentalness = {xnonpivotal}')
```

```
pivotal interval for instrumentalness = (-0.0003530405865580701, 0.00987983787295807)
non-pivotal interval for instrumentalness = (0.000521351685, 0.010774570615000003)
```

In the end let's take a look at liveness and see the relationship between global mean and mean of second cluster.

```
In [19]: x = np.random.choice(truncated[:, 3][prediction_5==2], (5000, 100)).mean(1)
         g = np.random.choice(truncated[:, 3], (5000, 100)).mean(1)
         d = x - g
         q = x / g

         fig, ax = plt.subplots(1, 2)
         ax[0].hist(d, bins=50);
         ax[0].set_title('m2 - m')
         ax[1].hist(q, bins=50);
         ax[1].set_title('m2/m');
```




```
In [20]: dpivotal, dnonpivotal = intervals(d)

         print(f'pivotal interval for d = {dpivotal}')
         print(f'non-pivotal interval for d = {dnonpivotal}')
```

```
pivotal interval for d = (-0.014699099224811829, 0.030452517224811834)
non-pivotal interval for d = (-0.014574224999999977, 0.03014522499999996)
```

```
In [21]: qpivotal, qnonpivotal = intervals(q)

         print(f'pivotal interval for q = {qpivotal}')
         print(f'non-pivotal interval for q = {qnonpivotal}')
```

```
pivotal interval for q = (0.9009191291853411, 1.2096099880330384)
non-pivotal interval for q = (0.9087577989079509, 1.2149678682574179)
```

3 Homework 3

```
In [22]: fig, ax = plt.subplots(nrows=1, ncols=2)

         ax[0].hist(df["valence"], bins=20);

         ax[1].hist(df["energy"], bins=20);
```

Takinh in count distribution of features, I propose following borders: * 0.35, 0.79 for valence * 0.4, 0.55, 0.7 for energy

Third feature is binary: mode.

As a main feature we will choose energy.

```
In [23]: nominals = np.zeros((100, 3), dtype=int)
         nominals[:, 0] = df['mode'].values

         for border in [0.35, 0.79]:
              nominals[:, 1] += df.valence.values > border

         for border in [0.4, 0.55, 0.7]:
              nominals[:, 2] += df.energy.values > border
```

```
In [24]: def analyze(x, y, n, m, x_prefix, y_prefix):
             def visualize(caption):
                 df = pd.DataFrame(
                     vis,
                     index=[f'{x_prefix}_{i}' for i in range(n)] + ['total'],
                     columns=[f'{y_prefix}_{i}' for i in range(m)] + ['total'],
                 )
                 display(df.style.set_caption(caption))

             data = np.zeros((n + 1, m + 1))
             for a, b in zip(x, y):
                 data[a, b] += 1
             data[-1, :-1] = data[:-1, :-1].sum(0)
             data[:-1, -1] = data[:-1, :-1].sum(1)
             data[-1, -1] =  data[:-1, :-1].sum()

             vis = data.copy()

             cond = data[:-1, :-1].copy()
             cond /= cond.sum(0)

             vis[:-1, :-1] = cond.copy()
             visualize('Conditional frequency')

             glob = data[:-1, :-1].copy()
             glob /= glob.sum()

             vis[:-1, :-1] = glob.copy()
             vis[-1, :-1] = vis[:-1, :-1].sum(0)
             vis[:-1, -1] = vis[:-1, :-1].sum(1)
             vis[-1, -1] =  vis[:-1, :-1].sum()
             visualize('Relative frequency')
             vis = data.copy()

             result = np.divide(cond, glob.sum(1).reshape(-1, 1)) - 1
             vis[:-1,:-1] = result
             visualize('Quetlet relative index')

             vis[:-1, :-1] = glob * result
             vis[-1, :-1] = vis[:-1, :-1].sum(0)
             vis[:-1, -1] = vis[:-1, :-1].sum(1)
             vis[-1, -1] =  vis[:-1, :-1].sum()
             visualize('Quetlet Summary')

In [25]: analyze(nominals[:, 2], nominals[:, 1], 4, 3, 'energy', 'valence')
```

Conditional frequency

	valence_0	valence_1	valence_2	total
energy_0	0.0	0.043478	0.000000	3.0
energy_1	0.4	0.173913	0.090909	21.0
energy_2	0.4	0.318841	0.454545	35.0
energy_3	0.2	0.463768	0.454545	41.0
total	20.0	69.000000	11.000000	100.0

Relative frequency

	valence_0	valence_1	valence_2	total
energy_0	0.00	0.03	0.00	0.03
energy_1	0.08	0.12	0.01	0.21
energy_2	0.08	0.22	0.05	0.35
energy_3	0.04	0.32	0.05	0.41
total	0.20	0.69	0.11	1.00

Quetlet relative index

	valence_0	valence_1	valence_2	total
energy_0	-1.000000	0.449275	-1.000000	3.0
energy_1	0.904762	-0.171843	-0.567100	21.0
energy_2	0.142857	-0.089027	0.298701	35.0
energy_3	-0.512195	0.131142	0.108647	41.0
total	20.000000	69.000000	11.000000	100.0

Quetlet Summary

	valence_0	valence_1	valence_2	total
energy_0	-0.000000	0.013478	-0.000000	0.013478
energy_1	0.072381	-0.020621	-0.005671	0.046089
energy_2	0.011429	-0.019586	0.014935	0.006778
energy_3	-0.020488	0.041965	0.005432	0.026910
total	0.063322	0.015237	0.014696	0.093255

We got nothing really meaningful from frequnecies tables, but there is interesting value of Quetlet index for valence '0' and energy '1' = 0.904762.

Since $(k-1)(l-1) = 6$ we got critical value equal to 12.592 for 95% confidence level and 16.812 for 99%. $N\chi^2 = 100 * 0.0932547 = 9.32547$ That means that our features are independent for both confidence levels.

Let's find suffice number of observations required to see features as associated.

```
In [ ]: N_95 = int(np.ceil(12.592 / 0.0932547))
        N_99 = int(np.ceil(16.812 / 0.0932547))
        print(f'To reject independence hypothesis we need'
              '{N_95} objects for 95% confidence and {N_99} for 99%')
```

```
To reject independence hypothesis we need 136 objects for 95% confidence and 181 for 99%
```

```
In [27]: analyze(nominals[:, 2], nominals[:, 0], 4, 2, 'energy', 'mode')
```

Conditional frequency

	mode_0	mode_1	total
energy_0	0.023810	0.034483	3.0
energy_1	0.166667	0.241379	21.0
energy_2	0.380952	0.327586	35.0
energy_3	0.428571	0.396552	41.0
total	42.000000	58.000000	100.0

Relative frequency

	mode_0	mode_1	total
energy_0	0.01	0.02	0.03
energy_1	0.07	0.14	0.21
energy_2	0.16	0.19	0.35
energy_3	0.18	0.23	0.41
total	0.42	0.58	1.00

Quetlet relative index

	mode_0	mode_1	total
energy_0	-0.206349	0.149425	3.0
energy_1	-0.206349	0.149425	21.0
energy_2	0.088435	-0.064039	35.0
energy_3	0.045296	-0.032801	41.0
total	42.000000	58.000000	100.0

Quetlet Summary

	mode_0	mode_1	total
energy_0	-0.002063	0.002989	0.000925
energy_1	-0.014444	0.020920	0.006475
energy_2	0.014150	-0.012167	0.001982
energy_3	0.008153	-0.007544	0.000609
total	0.005795	0.004196	0.009991

Well, features look completely independent according to our tables.

Since (k−1)(l−1)=3 we got critical value equal to 7.815 for 95% confidence level and 11.345 for 99%. $N\chi^2 = 100 * 0.00999144 = 0.999144$ That means that our features are independent for both confidence levels.

Let's find suffice number of observations required to see features as associated.

```
In [ ]: N_95 = int(np.ceil(7.815 / 0.00999144))
        N_99 = int(np.ceil(11.3452 / 0.00999144))
        print(f'To reject independence hypothesis we need'
              '{N_95} objects for 95% confidence and {N_99} for 99%')
```

```
To reject independence hypothesis we need 783 objects for 95% confidence and 1136 for 99%
```

4 Homework 4

We will use the same features as in the second homework. They are good at expressing mood of song.

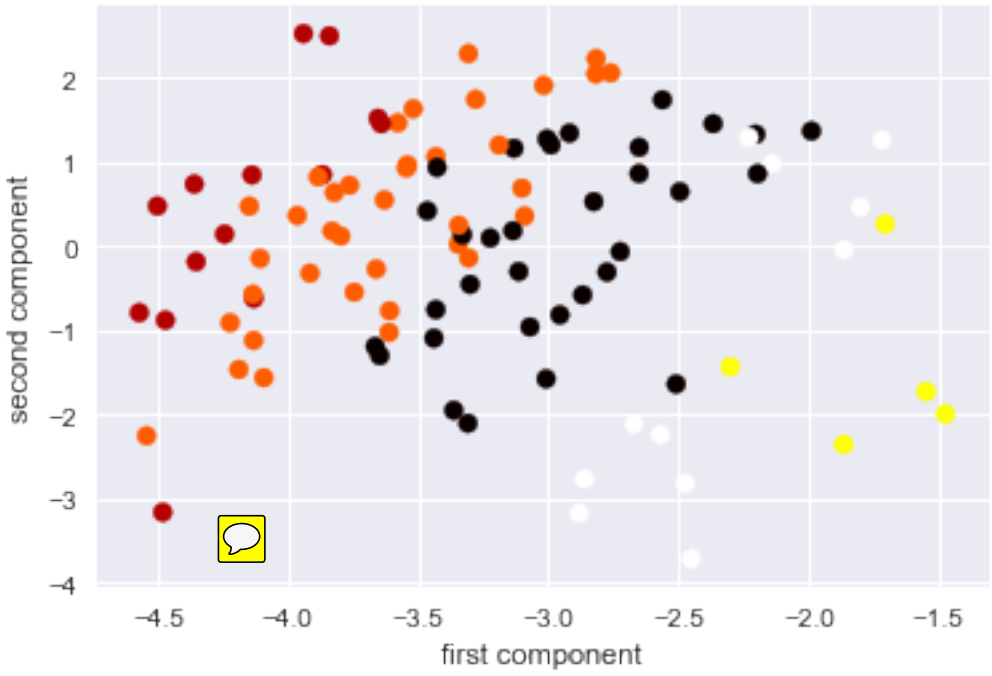
We will multiply all features by 100 (since most of them are from 0 to 1) and standartize loudness to fit in range $[0, 100]$.

```
In [ ]: standartized = 100 * truncated
        standartized[:, 1] = 100 * (standartized[:, 1] - standartized[:, 1].min()) /
        (standartized[:, 1].max() - standartized[:, 1].min())

        u, s, vt = np.linalg.svd(standartized)
```

```
In [ ]: for i, mu in enumerate(s):
            contrib = mu**2 / (standartized * standartized).sum()
            print('contribution of #{} component is {:.4} or {:.3}%'.format(i, contrib, contrib))
```

```
contribution of #0 component is 0.9404 or 0.94%
contribution of #1 component is 0.02808 or 0.0281%
contribution of #2 component is 0.01836 or 0.0184%
contribution of #3 component is 0.008329 or 0.00833%
contribution of #4 component is 0.004344 or 0.00434%
contribution of #5 component is 0.0004854 or 0.000485%
```

```
In [31]: plt.scatter(u[:, 0]*np.sqrt(s[0]), u[:, 1]*np.sqrt(s[1]), c=prediction_5, cmap='hot')
         plt.xlabel('first component')
         plt.ylabel('second component');
```



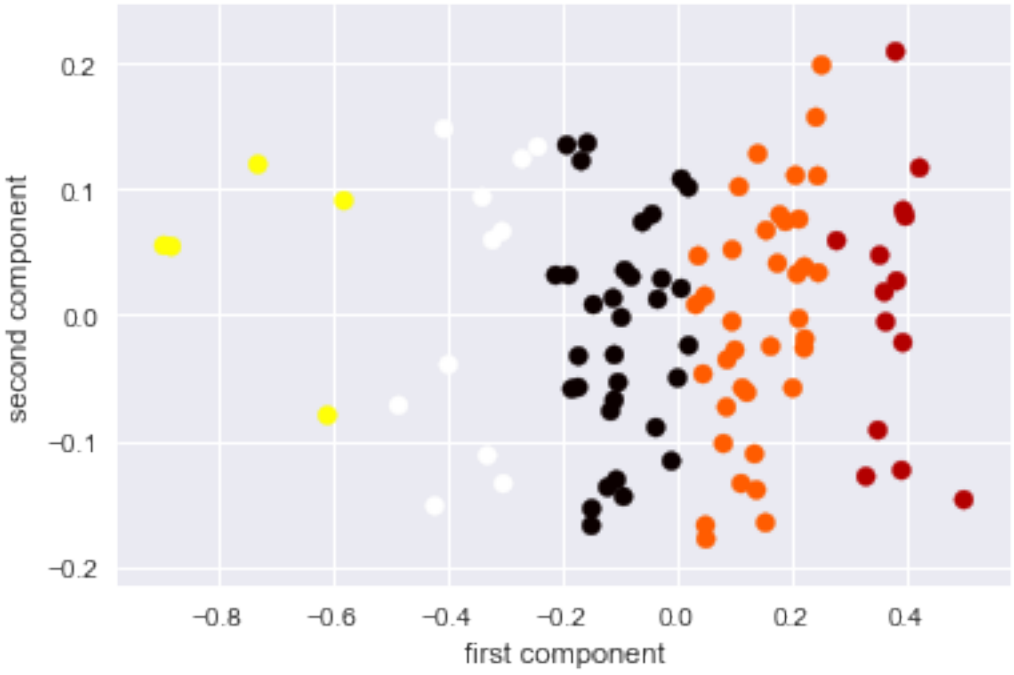
Now we will centralize and divide by std.

```
In [32]: standartized = (truncated - truncated.mean(0)) / truncated.std()
         u, s, vt = np.linalg.svd(standartized)
```

```
In [33]: for i, mu in enumerate(s):
             contrib = mu**2 / (standartized * standartized).sum()
             print('contribution of #{} component is {:.4} or {:.3}%'.format(
             i, contrib, contrib))
```

```
contribution of #0 component is 0.9753 or 0.975%
contribution of #1 component is 0.01259 or 0.0126%
contribution of #2 component is 0.007435 or 0.00743%
contribution of #3 component is 0.002706 or 0.00271%
contribution of #4 component is 0.001746 or 0.00175%
contribution of #5 component is 0.000191 or 0.000191%
```

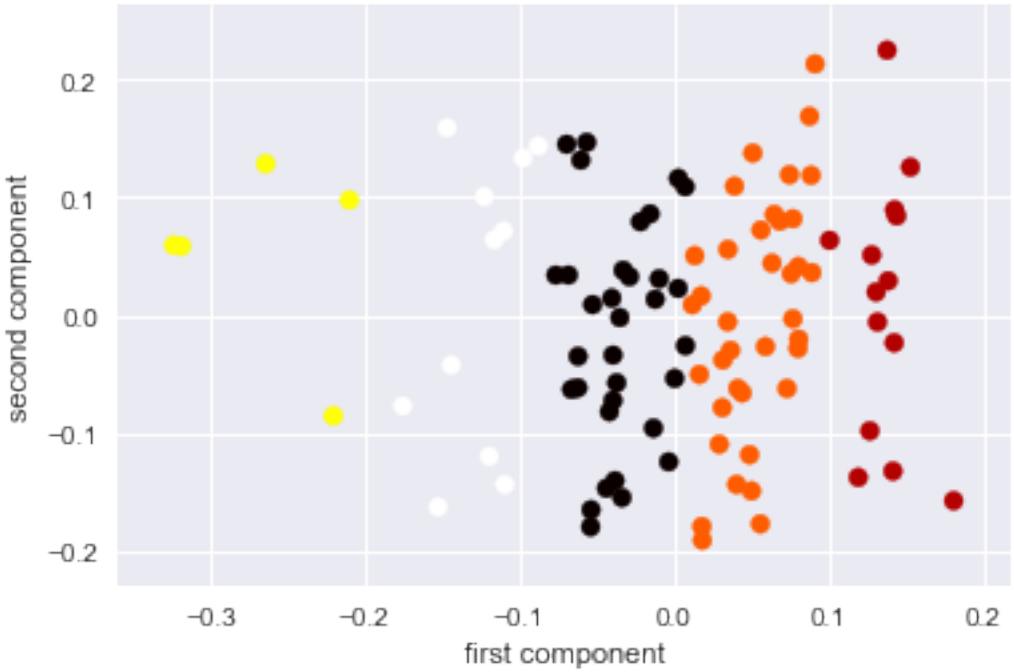
```
In [34]: plt.scatter(u[:, 0]*np.sqrt(s[0]), u[:, 1]*np.sqrt(s[1]), c=prediction_5, cmap='hot')
         plt.xlabel('first component')
         plt.ylabel('second component');
```



Hey! Looks much better. This standardization makes features all be treated as equally important. Since PCA will treat variables with greater variance as more important, because it tries to account for the greatest possible total variance with each successive component.

Let's make conventional PCA with same standartization.

```
In [35]: Y = standartized

In [36]: B = Y.T.dot(Y) / Y.shape[0]

In [ ]: def pc(Y, B):
            La, C = np.linalg.eig(B)
            max_index = np.argmax(La)
            return Y.dot(C[:, max_index]) / np.sqrt(Y.shape[0] * La[max_index]),\
                C[:, max_index], La[max_index]

In [38]: z, c, l = pc(Y, B)

In [39]: B_ = B - l * c[:, None].dot(c[:, None].T)

In [40]: z_, c_, l_ = pc(Y, B_)

In [41]: plt.scatter(z, -z_, c=prediction_5, cmap='hot')
         plt.xlabel('first component')
         plt.ylabel('second component');
```



The same output! Well, it is definitely what we expected, since the underlying math is the same.

Now let's use standartized dataset with all features ranked from 0 to 100. Then we'll apply PCA and describe this hidden factor.

```
In [ ]: standartized = 100 * truncated
        standartized[:, 1] = 100 * (standartized[:, 1] - standartized[:, 1].min())\
        / (standartized[:, 1].max() - standartized[:, 1].min())
        u, s, vt = np.linalg.svd(standartized)
```

Values of hidden factor:

```
In [43]: standartized.dot(vt[0]) / vt[0].sum()
```

```
Out[43]: array([75.5505317 , 71.1839614 , 69.66124484, 47.61500348, 66.00931653,
                44.68376952, 46.76158846, 64.01478435, 60.84457908, 28.93027549,
                50.64718964, 43.17145094, 55.7215492 , 63.13979415, 30.33436545,
                64.42866374, 71.54466539, 60.31976726, 70.61586296, 66.83167661,
                50.8631513 , 55.7389374 , 48.57662538, 76.59793412, 52.79191037,
                42.27847813, 75.89841082, 36.06480462, 55.77426409, 56.39543903,
                61.48159236, 48.33841759, 65.19867771, 69.65045692, 56.12607392,
                45.01705767, 26.07715211, 37.11828721, 49.81155723, 55.63136567,
                37.00761067, 69.93627028, 73.38741378, 63.44261485, 69.70431956,
                56.69135778, 24.82565882, 77.0709575 , 46.52120451, 52.10071812,
                66.43216176, 38.77573367, 47.4626942 , 59.76566828, 49.19244061,
                69.76609883, 69.23798746, 57.97670155, 31.41972516, 65.47468139,
                42.03967514, 61.78065479, 57.85160571, 57.84298596, 56.36446955,
                51.7522893 , 28.74019737, 50.7042785 , 73.5051653 , 44.67249704,
                47.48363776, 61.59586101, 45.9047022 , 37.56331633, 52.85969764,
                61.38725142, 41.2950421 , 64.74599097, 52.47906768, 55.27182794,
                60.88368389, 75.37811543, 61.71609638, 58.40858331, 64.57856725,
                43.30034619, 39.88473481, 59.69096157, 50.3889394 , 59.31898238,
                41.69346227, 69.00272892, 54.3255402 , 61.18582759, 31.38752549,
                53.74372706, 57.77724339, 48.21451755, 52.27296692, 33.51283711])
```

```
In [44]: for name, coeff in zip(columns, vt[0]):
             print(f'{name}: {coeff}')
```

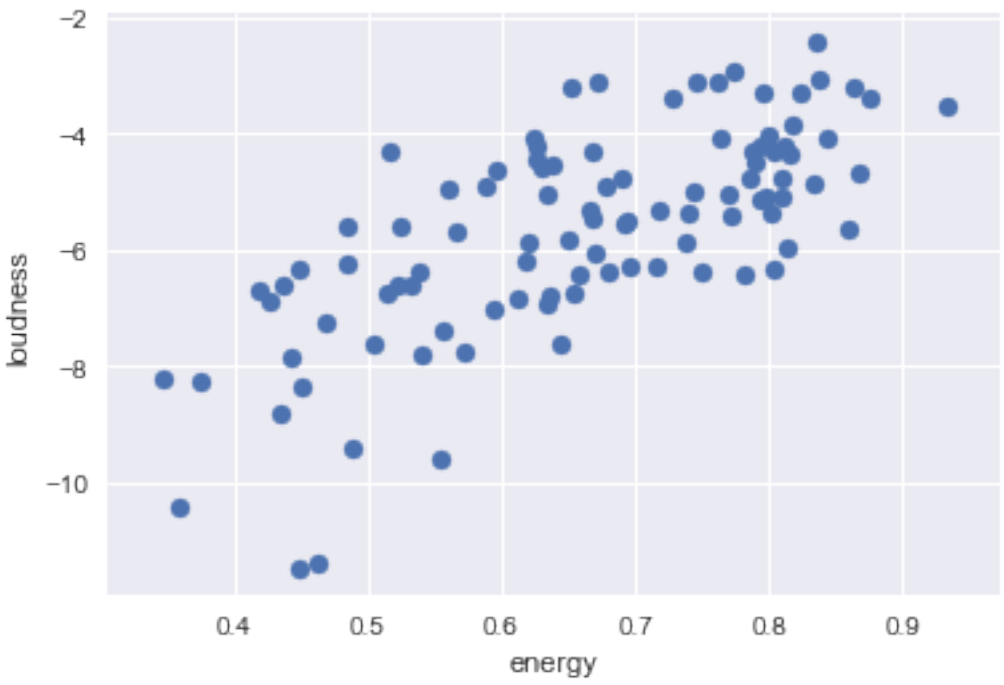
```
energy: -0.6036980274946596
loudness: -0.5990205362431725
acousticness: -0.1459306476983379
liveness: -0.1342568325297601
instrumentalness: -0.004074823083509584
valence: -0.4872225708608505
```

So, as we mentioned before, those features represent the mood of given song, and the hidden factor depends mostly on energy, loudness and valence. Louder, more energetic songs are more cheerful, otherwise they are count as sad.

4.1 Homework 5

Let's take a look at energy and loudness.

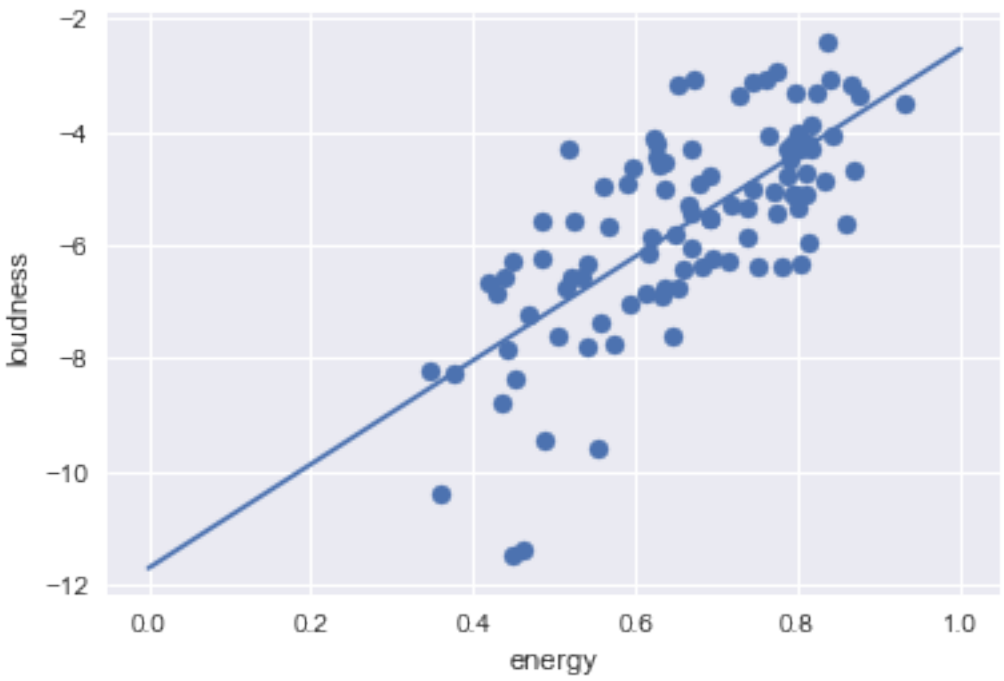
```
In [45]: plt.scatter(df.energy, df.loudness)
         plt.xlabel('energy')
         plt.ylabel('loudness');
```




```
In [46]: y = df.loudness.values
         x = df.energy.values

In [47]: a = ((x-x.mean())*(y-y.mean())).mean() / x.var()
         b = y.mean() - a * x.mean()
         a, b

Out[47]: (9.17998799563441, -11.717776268835696)
```

So the slope means that for each change of energy by 0.1 loudness increases by 0.9.

```
In [48]: z = np.linspace(0, 1, 100)
         t = [a*x + b for x in z]

         plt.scatter(x, y)
         plt.plot(z, t)
         plt.xlabel('energy')
         plt.ylabel('loudness');
```



Calculations of correlation and detrminacy coefficients.

```
In [49]: corr_coef = a * x.std() / y.std()

In [50]: determ_coef = corr_coef**2

In [51]: corr_coef, determ_coef

Out[51]: (0.7091414022403669, 0.5028815283714339)
```

Half of variance is described by our model, which is acceptable, and we may say that features are correlated.

Let's make a few predictions according to our model.

```
In [52]: order = np.argsort(x)

In [ ]: print(f'predictions for {y[order][3::33]}:'
              '{a * x[order][3::33] + b} at points {x[order][3::33]}')

predictions for [-6.678 -4.2   -5.043]: [-7.88054129 -5.97110378 -4.6583655 ]
at points [0.418 0.626 0.769]
```

At the low values we have a relatively huge error, but in the middle and at high values of energy predictions become more accurate.

Now, we will compute MAE and RAE.

```
In [54]: mean_absolute_error = (np.abs(y - (a * x + b))).mean()
         mean_absolute_error

Out[54]: 0.9888019479844984

In [55]: relative_absolute_error = (np.abs(y - (a * x + b))).sum() / np.abs(y-y.mean()).sum()
         relative_absolute_error

Out[55]: 0.7098525938747324
```

In average we make a mistake of 1 dB. And our prediction is in 1.43 times better than a constant prediction. Combining our knowledge about determinacy, RAE and MAE we can conclude that our prediction isn't very accurate and gives a low-quality dependance between theese two factors.