Clustering of Songs with Amazon Web Services

Certification project for AIDA 2020

Falk Lutz
Julian Godley
Simon Harston

Agenda

- Project Tasks
- Approach
- Data Preprocessing
- Exploratory Data Analysis
- Clustering Algorithms
- Distributed Cluster-Computing Frameworks
- Findings

- Project Tasks - Status

DFKI project description:

Project Tasks

The final goal is to find out new songs that can be part of your list of favourite songs. To accomplish this task, different ways of using a clustering approach are going to be compared. On the one hand, through a clustering approach following a distributed cluster-computing framework, and on the other hand, through the use of Amazon RDS. To do that, some tasks need to be accomplished:

1. Understand the content that is available in the dataset and clean the dataset if it is necessary.
2. Clustering following a distributed cluster-computing framework.
    a. Choose the best clustering algorithm taking into account the attributes that the dataset offers. In order to decide the best approach, it could be useful to have an overview of the different perspectives explaining the advantages and drawbacks of your decision.
    b. Evaluate your model statistically.
3. Clustering in the cloud using AWS.
    a. Create a database using Amazon RDS.
    b. Create tables to load the data of the dataset.
    c. Export the data into S3 and apply a cluster analysis in AWS using a different clustering algorithm.
    d. Analyze the outcomes.
    e. (Optional task) Put the final results into RDS for future access.
4. Use the different packages of visualization explained during the course to visualize clusters based on your findings.
5. Compare the findings of both methods using metrics, the visualizations, etc.

Approach

Agile, iterative approach defining small daily deliverables and thereby gaining insight and improvements

Team members assumed individual pipeline tasks and determined clear handover points.

Collaborative decision making enabled rapid adaptation of previously generated output in the areas of:

- EDA
- Data Preprocessing
- Output facilitation on RDS

Data Preprocessing

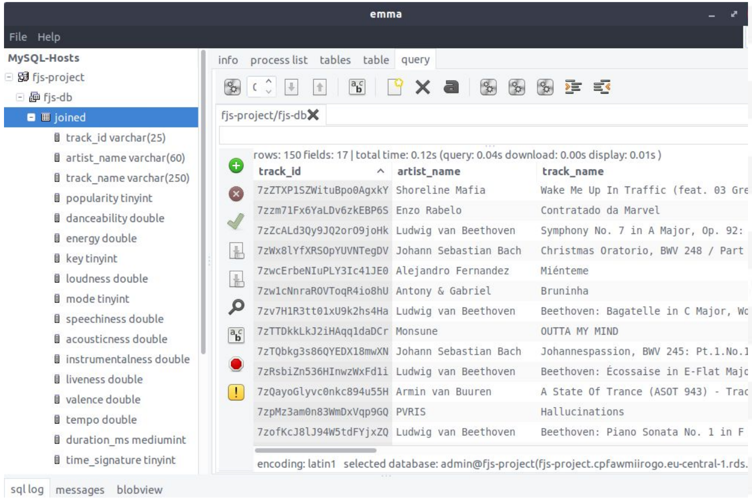
- We received music title data in 3 CSV-file datasets - ‘world’: 9,320 rows, ‘brazil’: 9,239 rows, ‘2020’: 1,742 rows.
- These were stored on our S3-bucket and retrieved from there by our Sagemaker notebooks.
- After concatenating all rows (= 20,301 rows) we removed exact (-4481 = 15,820 rows) and high-similarity duplicates (-194 = 15,626 rows).

```python
# Now some more complicated duplicates - same artist, track_name and duration

subset = 'artist_name track_name duration_ms'.split()

df_joined[df_joined.duplicated(subset=subset, keep=False)].sort_values(subset)
```

	artist_name	track_name	track_id	popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
4268	As I Lay Dying	Blinded	5xa5C8TmCuF1q8cBRutiMY	62	0.339	0.9720	8	-5.260	1	0.1850	0.000006	0.145000	0.3650	0.1250	200.113	202158
8774	As I Lay Dying	Blinded	2HdjEa5BP2VACt1velDTIk	56	0.332	0.9720	8	-5.258	1	0.1980	0.000006	0.141000	0.3650	0.1220	200.153	202158
3911	As I Lay Dying	My Own Grave	6bDoeNLCA32SYhTzIk5w5y	61	0.475	0.9940	5	-4.978	0	0.2500	0.000113	0.012700	0.0535	0.0399	125.033	253318
6751	As I Lay Dying	My Own Grave	0CcqWuAEJC93K8cBMbAjgI	57	0.477	0.9940	5	-4.953	0	0.2410	0.000115	0.012900	0.0534	0.0397	125.058	253318
3110	Bastille	Admit Defeat	1OHAFggB1Jatvto7HvUN4L	56	0.653	0.6410	5	-7.007	1	0.0475	0.155000	0.000000	0.2070	0.4730	90.036	185680
1952	Bastille	Admit Defeat	4qcnL8k4i8Ynw7kAPgVoD6	54	0.651	0.6400	5	-7.019	1	0.0460	0.149000	0.000000	0.2010	0.4920	90.044	185680
6764	Cigarettes After Sex	Cry	0r0zdaQ9S3fouDnvJ25pwl	63	0.408	0.3970	7	-10.452	1	0.0279	0.756000	0.658000	0.1120	0.1760	142.668	256800
8581	Cigarettes After Sex	Cry	0Qr61NXlyAeQaADO5xn3rl	53	0.409	0.3990	7	-10.456	1	0.0275	0.763000	0.654000	0.1120	0.1610	142.823	256800

Data Preprocessing

- We received music title data in 3 CSV-file datasets - ‘world’: 9,320 rows, ‘brazil’: 9,239 rows, ‘2020’: 1,742 rows.
- These were stored on our S3-bucket and retrieved from there by our Sagemaker notebooks.
- After concatenating all rows (= 20,301 rows) we removed exact (-4481 = 15,820 rows) and high-similarity duplicates (-194 = 15,626 rows).
- In all 14 feature columns, there were no null values.
- The joined dataset was written to a CSV-file on our S3 bucket and to a table on an RDS database instance.



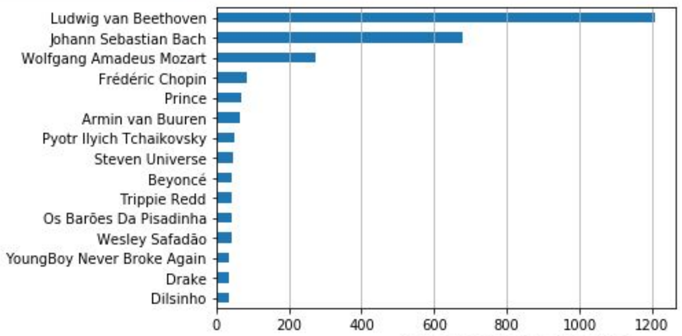

```
df_joined.describe().rename_axis('column').T.astype({'count':int})
```

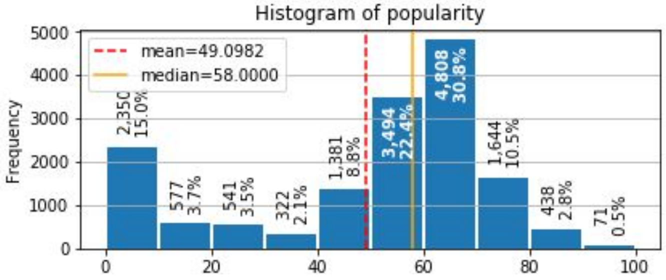
column	count	mean	std	min	25%	50%	75%	max
popularity	15626	49.098234	24.273116	0.00000	43.00000	58.000000	65.000000	100.000
danceability	15626	0.594562	0.195956	0.00000	0.45800	0.622000	0.745000	0.983
energy	15626	0.534537	0.271949	0.00002	0.33700	0.588000	0.750000	1.000
key	15626	5.183348	3.594108	0.00000	2.00000	5.000000	8.000000	11.000
loudness	15626	-10.063025	7.372410	-45.13600	-11.83775	-7.164500	-5.221000	2.036
mode	15626	0.633239	0.481936	0.00000	0.00000	1.000000	1.000000	1.000
speechiness	15626	0.109787	0.118288	0.00000	0.03990	0.056200	0.125000	0.951
acousticness	15626	0.417139	0.362938	0.00000	0.07410	0.309000	0.783000	0.996
instrumentalness	15626	0.167178	0.333673	0.00000	0.00000	0.000003	0.014075	0.998
liveness	15626	0.200374	0.184837	0.00000	0.09760	0.123000	0.227000	1.000
valence	15626	0.475397	0.250612	0.00000	0.27400	0.470500	0.671000	0.989
tempo	15626	119.823206	30.937673	0.00000	95.01500	120.143500	140.030750	235.998
duration_ms	15626	204291.707283	96346.332970	12564.00000	160112.00000	191793.000000	226003.000000	1493227.000
time_signature	15626	3.884423	0.502236	0.00000	4.00000	4.000000	4.000000	5.000

Exploratory Data Analysis

- Analysis of “artist” and “popularity” features showed a very strong bias to classical, especially Beethoven, music…
- And ~15% of all titles are very unpopular.

```
df_joined['artist_name'].value_counts()[15::-1].plot.barh().grid(axis='x')
```





Column	DType	Null	nUnique	Uniques [15626 rows total]	Top_10	Min	Max	Median	Mean
track_id	object	0 (0.0 %)	15626	['001UkMQHw4zXfFNdKpwXAF' '00314r7dPEVFJvETEjXKTH' '003VDDA7J3Xb2ZFINx7nIZ' ... '7zw1cNnraROVToqR4io8hU' '7zwcErbeNIuPLY3Ic41JE0' '7zzm71Fx6YaLDv6zkEBP6S']	{'2TyGVInCFqKCNflq2WcQZW': 1, '4n5KoOsuEBe4NCTu0jbtKt': 1, '2tXIRosY0nvmJvbrcQN7PS': 1, '25x38hVatSDa5ptfbKwdn5': 1, '5wbqhfN9s9Fbeyi5hHdn2W': 1, '6UnZTpSFCPuLWEZGWoqQmD': 1, '0dOeDvrzuSXSEujH48Fo4i': 1, '3CPP2MBrbPXH9FOaSTTyuO': 1, '7Lflb7ssXJ6hCcoMf5U1Wb': 1, '6zvxOpyf...	001UkMQHw4zXfFNdKpwXAF	7zzm71Fx6YaLDv6zkEBP6S	NaN	NaN
artist_name	object	0 (0.0 %)	4174	['NOT"uicideBoy$' '(G)I-DLE' ... 'ギヴン' '美波' '須田景凪']	{'Ludwig van Beethoven': 1206, 'Johann Sebastian Bach': 679, 'Wolfgang Amadeus Mozart': 275, 'Frédéric Chopin': 83, 'Prince': 68, 'Armin van Buuren': 64, 'Pyotr Ilyich Tchaikovsky': 50, 'Steven Universe': 48, 'Beyoncé': 44, 'Trippie Redd': 43}	$NOT	須田景凪	NaN	NaN
track_name	object	0 (0.0 %)	14876	['!' '"Es ist vollbracht", WoO 97' '"Nun komm, der Heiden Heiland", BWV 62: 1. Chorus "Nun komm, der Heiden Heiland"' ... '說好不哭' "Cosmic'.m4a' '달라달라 (DALLA DALLA)']	{'Silent Night': 9, 'Have Yourself A Merry Little Christmas': 8, 'Forever': 7, 'Home': 7, 'Cold': 6, 'Intro': 6, 'Feelings': 6, 'Sleigh Ride': 6, 'Someone You Loved': 6, 'Fire': 5}	!	달라달라 (DALLA DALLA)	NaN	NaN
popularity	int64	0 (0.0 %)	101	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 6...	{0: 938, 61: 633, 60: 625, 62: 618, 59: 575, 58: 572, 63: 539, 57: 502, 64: 499, 65: 445}	0	100	58	49.0982
danceability	float64	0 (0.0 %)	952	[0. 0.0566 0.0606 0.0608 0.0609 0.0618 0.0619 0.0624 0.0626 0.0629 0.063 0.0638 0.064 0.0644 0.0652 0.0657 0.0665 0.0695 0.0703 0.0709 0.0711 0.0714 0.0723 0.0724 0.0726 0.0727 0.0737 0.0748 0.0749 0.075 0.0771 0.0772 0.0775 0.0777 0.0778 0.078 0.0787 0.0789 0.079...	{0.669: 47, 0.747: 46, 0.745: 45, 0.774: 44, 0.737: 44, 0.731: 43, 0.722: 42, 0.726: 42, 0.623: 42, 0.7859999999999999: 40}	0	0.983	0.622	0.594562
energy	float64	0 (0.0 %)	1791	[2.02e-05 2.03e-05 1.39e-04 ... 9.97e-01 9.98e-01 1.00e+00]	{0.684: 39, 0.71: 38, 0.574: 37, 0.721: 37, 0.726: 37, 0.7979999999999999: 36, 0.696: 35, 0.63: 35, 0.664: 34, 0.6759999999999999: 34}	2.02e-05	1	0.588	0.534537
key	int64	0 (0.0 %)	12	[0 1 2 3 4 5 6 7 8 9 10 11]	{0: 1866, 1: 1700, 7: 1661, 2: 1556, 9: 1391, 5: 1390, 11: 1226, 10: 1110, 6: 1078, 8: 1016}	0	11	5	5.18335
					{-4.788: 8, -5.516: 8, -4.468999999999999: 8, -4.724: 8				

Exploratory Data Analysis

- Analysis of value ranges showed which features need scaling for use in clustering algorithms.
- Also, we looked for outliers per feature and quantified their volume in the data.

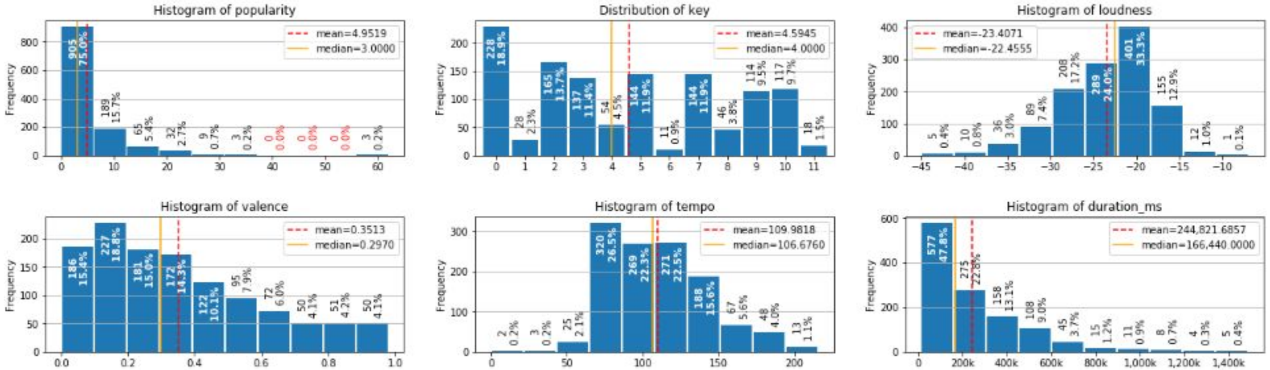
```
print(f'Total rows of data in df_joined dataframe: {len(df_joined):,}')

df_trimmed = df_joined.query('''(loudness < -35) or (loudness > 0)
    or (speechiness > 0.65) or (tempo < 1) or (duration_ms > 430000)
    or (time_signature < 1)'''.replace('\n', ''))

draw_hist_plots('df_trimmed')
```

```
Total rows of data in df_joined dataframe: 15,626
Total rows of data in df_trimmed dataframe: 609
```

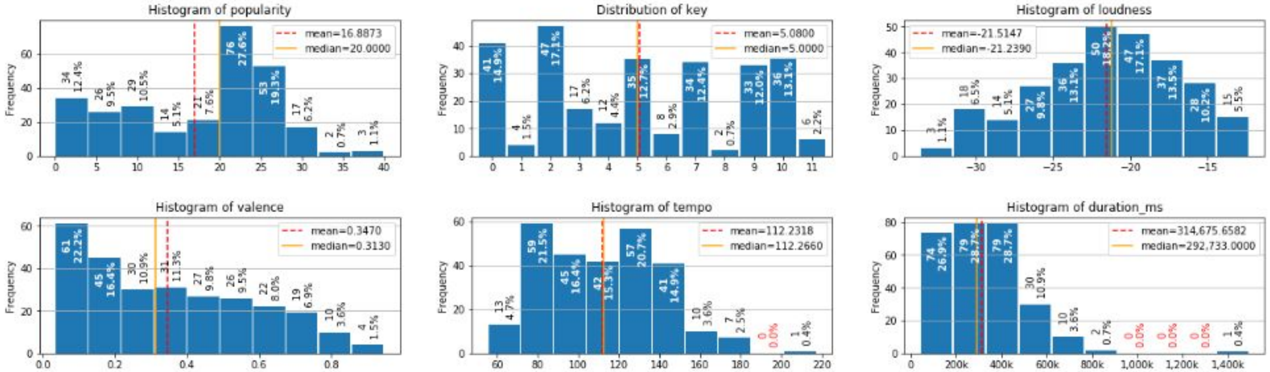
Exploratory Data Analysis

Some playful comparisons of data were performed...

Total rows of data in Beethoven dataframe: 1,206



Total rows of data in Mozart dataframe: 275

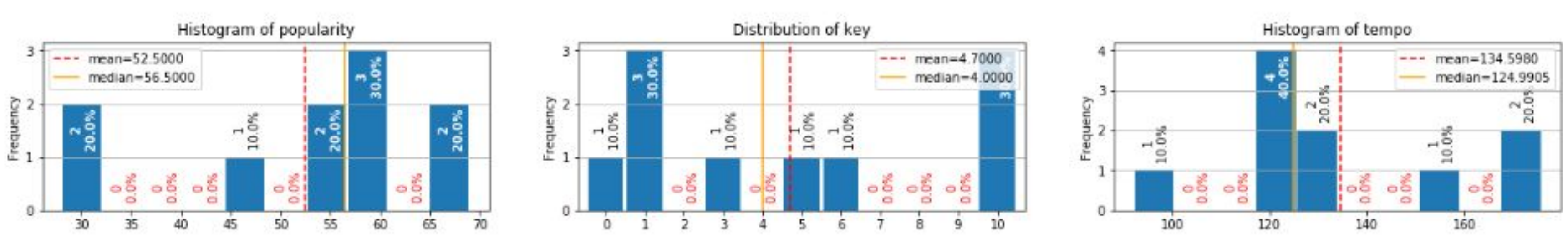


Exploratory Data Analysis

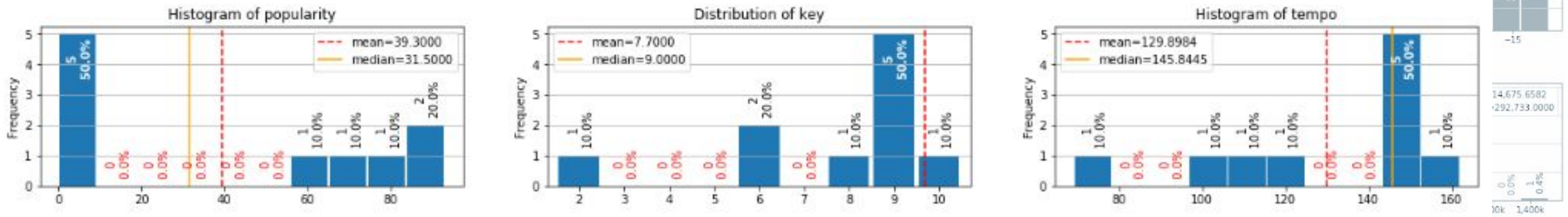
Some playful comparisons of data were performed...

```
Total rows of data in Beethoven dataframe: 1,206
```


```
Total rows of data in Madonna dataframe: 10
```

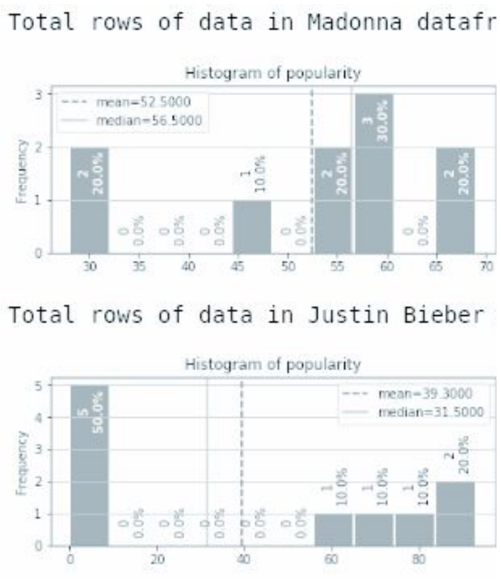


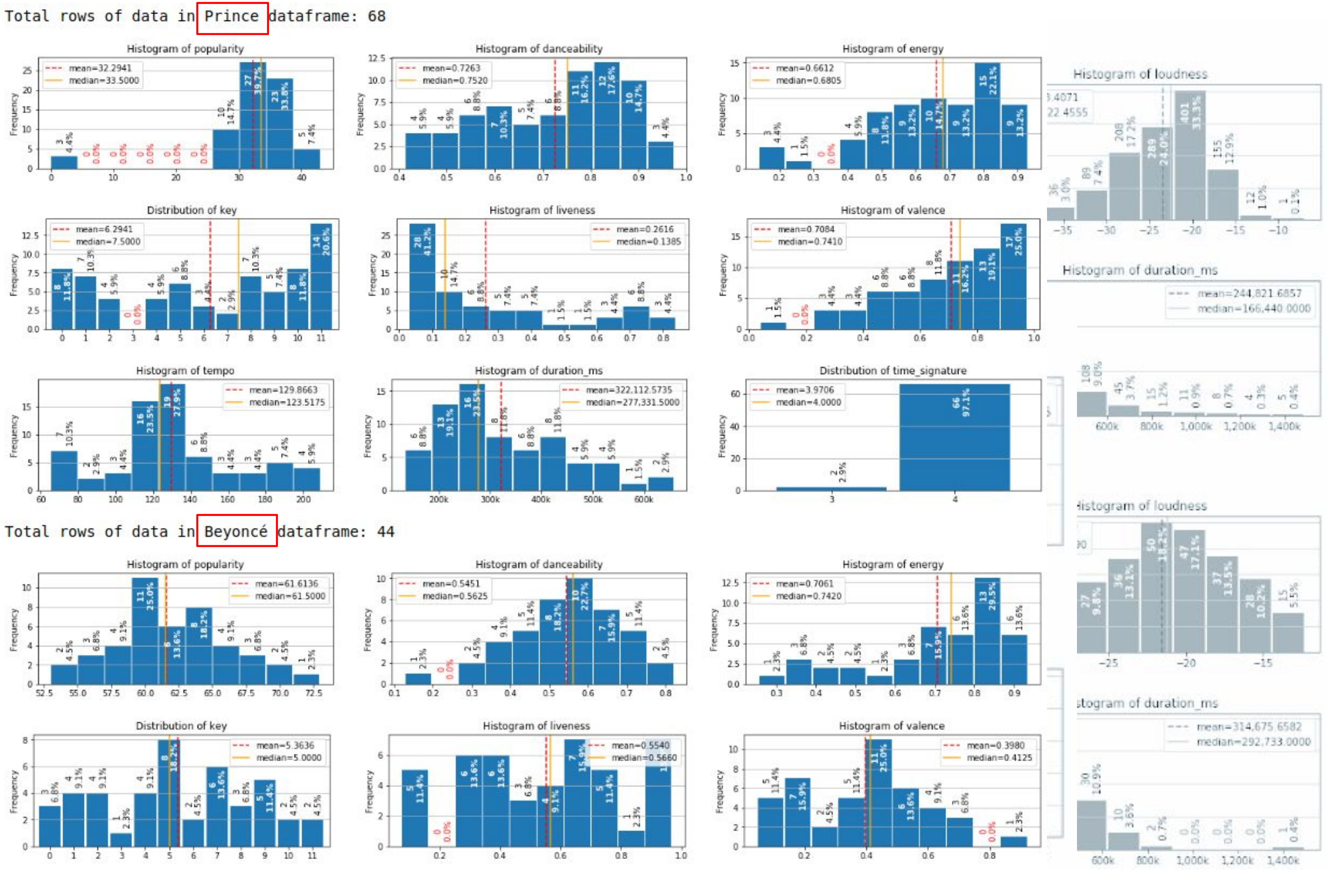
```
Total rows of data in Justin Bieber dataframe: 10
```



Exploratory Data Analysis

Some playful comparisons of data were performed...



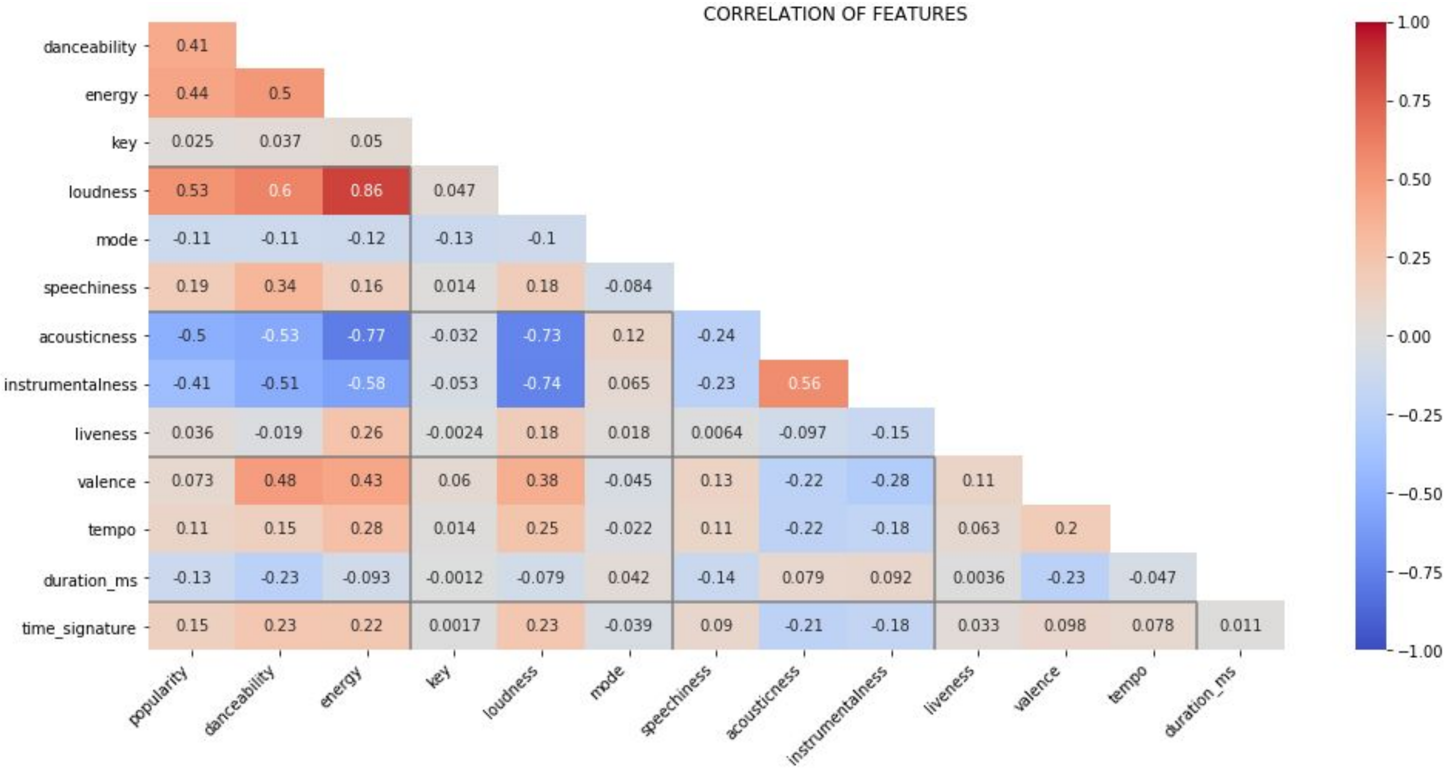


Exploratory Data Analysis

And feature correlation visualized.

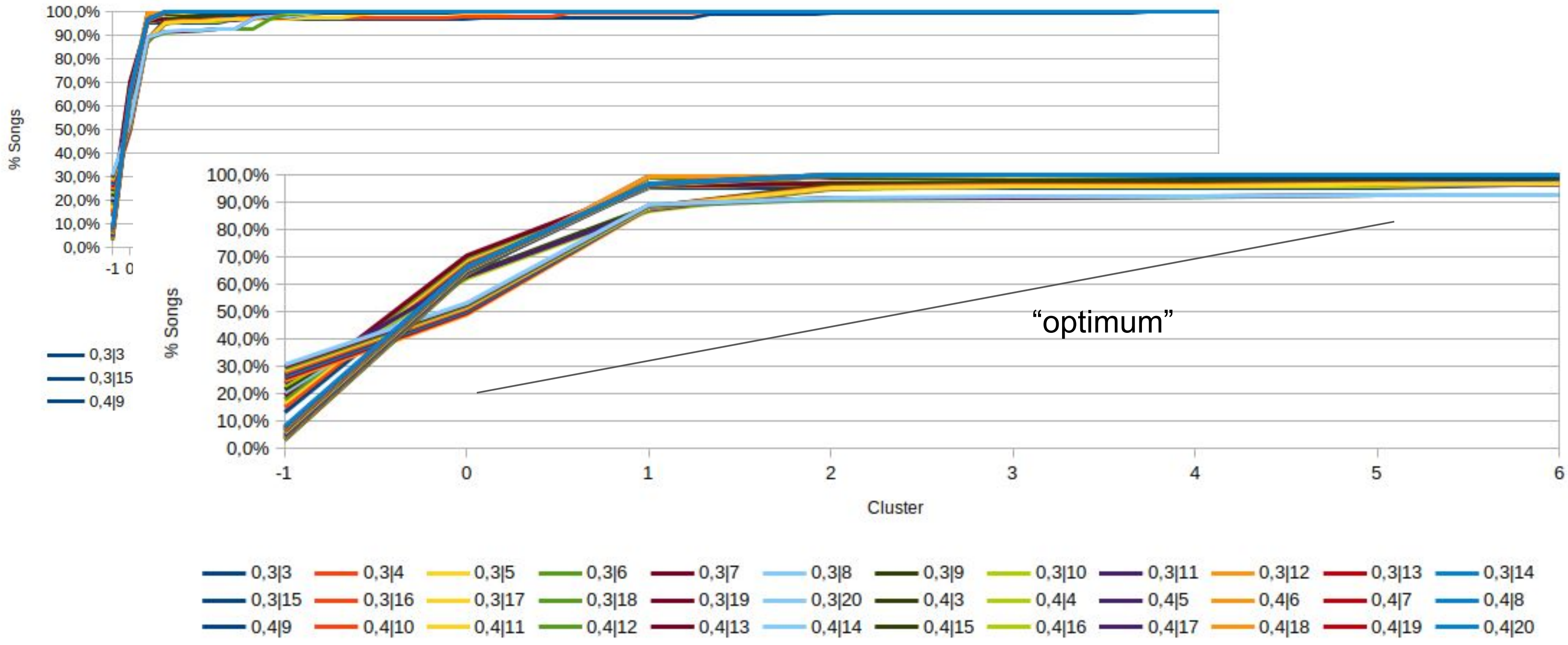
Notable findings:

- loud titles are full of positive energy!
- acoustic titles are generally neither loud nor energetic.



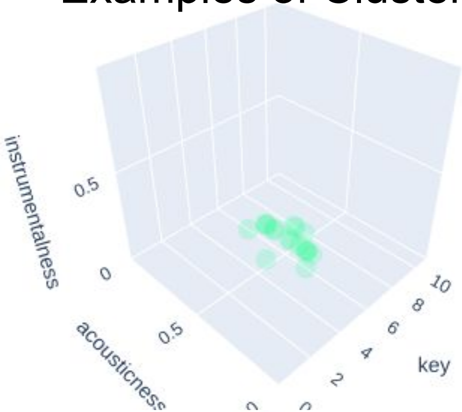
AWS Sagemaker Individual Notebook DBScan

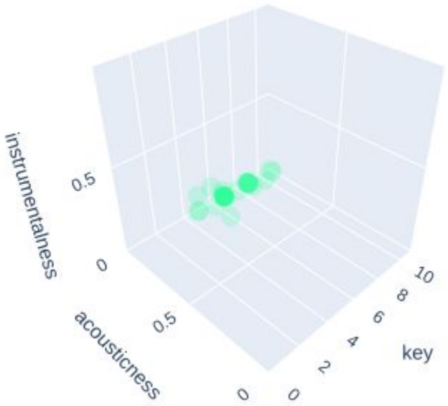
Hyperparameter Tuning

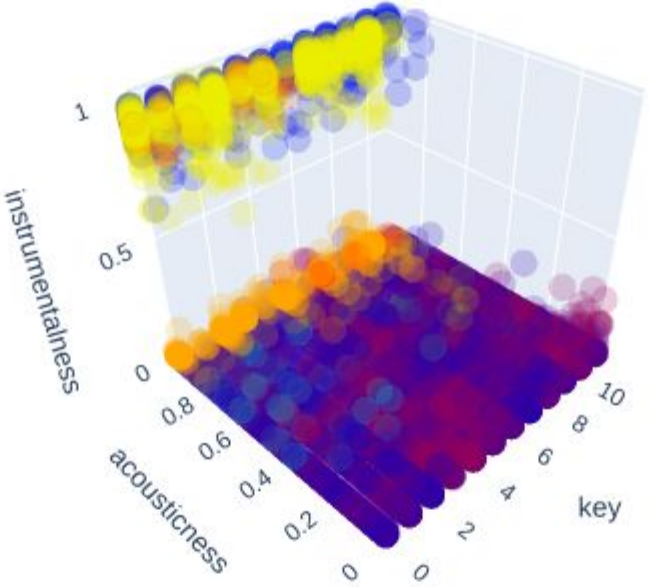


AWS Sagemaker standalone notebook DBScan

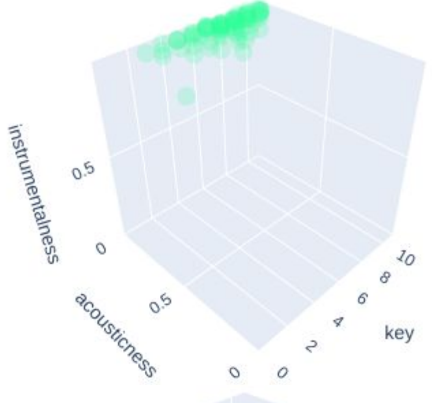
Examples of Cluster Visualizations

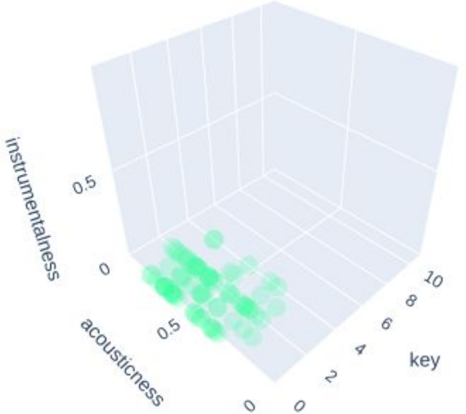






Cluster diagram for all Clusters (excl. "noise")

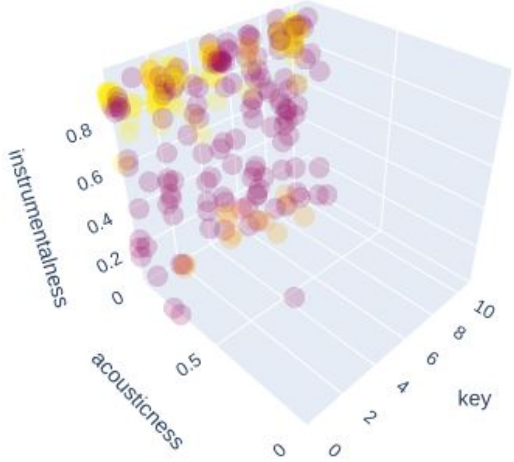


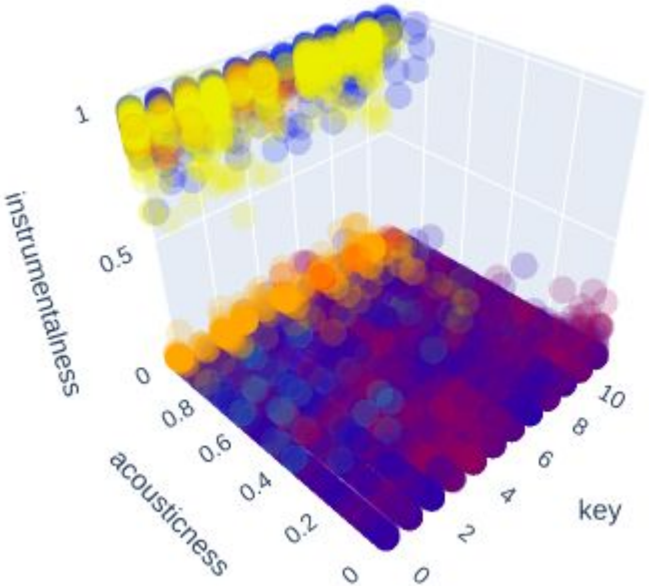


AWS Sagemaker standalone notebook DBScan

Examples of Artist Visualizations

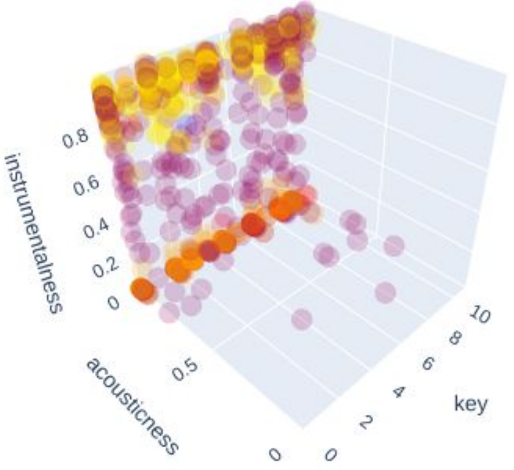
WAM



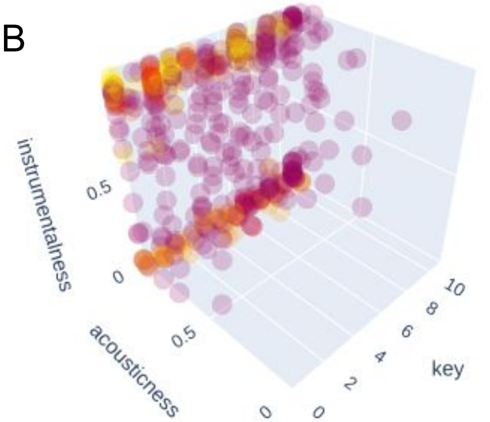


Cluster diagram for all Clusters (excl. "noise")

LvB



JSB



k-means clustering using the Amazon SageMaker Data science workflow

Amazon SageMaker

- Using built-in Amazon SageMaker algorithms
- Training jobs run on scalable AWS instances
- Data and artifact storage in AWS S3 bucket

```python
from sagemaker import KMeans
kmeans = KMeans(role=role,
train_instance_count=1,
train_instance_type='ml.c4.xlarge',
output_path=output_path,
k=24)
```

```
2020-12-03 21:19:35 Starting - Starting the training job...
2020-12-03 21:19:40 Starting - Launching requested ML instances......
2020-12-03 21:20:47 Starting - Preparing the instances for training......
2020-12-03 21:21:51 Downloading - Downloading input data...
2020-12-03 21:22:32 Training - Training image download completed. Training in progress.
```

```
2020-12-03 21:11:14 Uploading - Uploading generated training model
2020-12-03 21:11:14 Completed - Training job completed
Training seconds: 46
Billable seconds: 46
CPU times: user 779 ms, sys: 34.6 ms, total: 813 ms
Wall time: 3min 12s
```

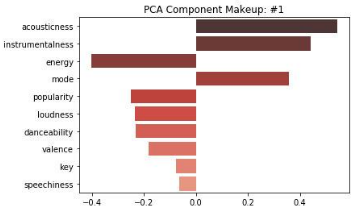
Amazon SageMaker > Models

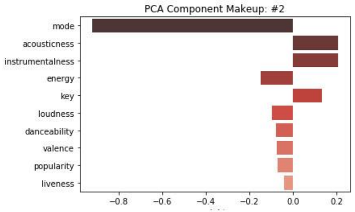
Models

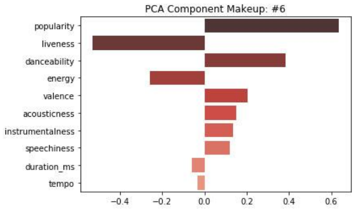
Search models

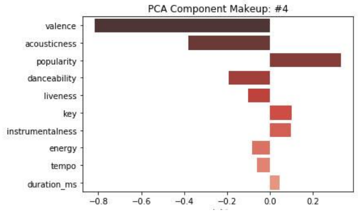
Name	ARN
kmeans-2020-12-03-21-23-17-478	arn:aws:sagemaker:eu-central-1:89
pca-2020-12-03-21-11-31-799	arn:aws:sagemaker:eu-central-1:89
kmeans-2020-12-03-19-42-46-824	arn:aws:sagemaker:eu-central-1:89
pca-2020-12-03-19-29-57-767	arn:aws:sagemaker:eu-central-1:89

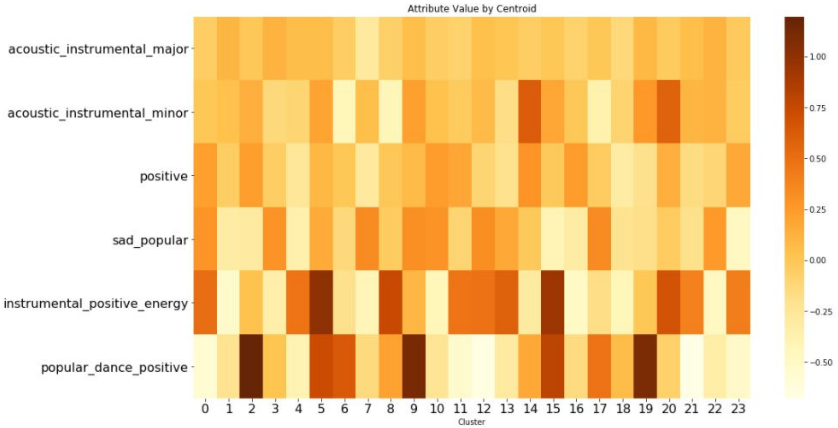
- Dimensionality reduction using principal components analysis (PCA)
- Data clustering using k-means
- Analyzing and interpreting clusters
- Making predictions for new songs

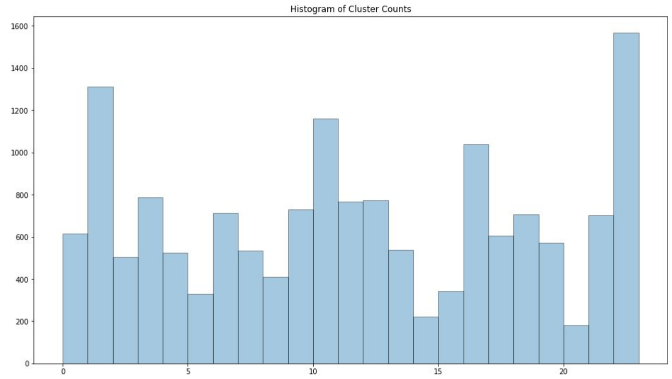






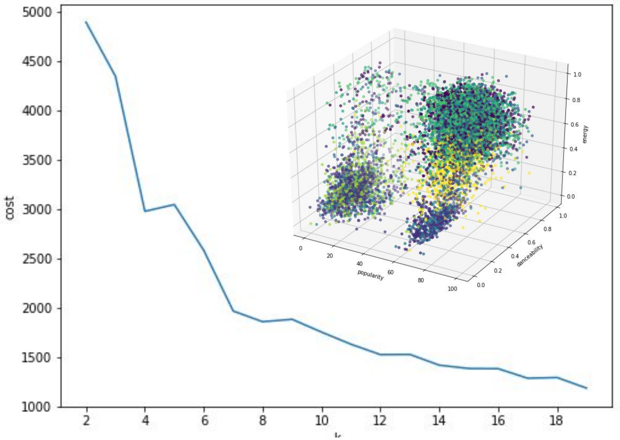






k-means clustering using a distributed cluster-computing framework

- Using Databricks platform for big data processing
- Using PySpark SQL Dataframes and built-in PySpark ML algorithms
- Implementation of Amazon RDS for reading and writing data



cluster	popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
0	91.060%	93.953%	78.392%	60.259%	81.424%	9.524%	55.091%	19.930%	16.805%	49.650%	86.740%	99.555%	8.072%	67.373%
1	23.933%	19.662%	15.867%	68.660%	29.358%	14.205%	3.636%	79.042%	67.330%	47.840%	23.913%	78.842%	43.947%	54.545%
2	22.826%	49.733%	15.214%	50.723%	8.173%	51.816%	39.070%	83.267%	68.363%	60.325%	87.454%	77.218%	1.683%	32.394%
3	46.099%	38.993%	43.266%	61.590%	49.257%	40.420%	10.146%	52.182%	40.415%	79.634%	46.180%	85.897%	26.822%	65.657%
4	91.185%	82.696%	87.460%	59.758%	90.614%	14.957%	33.912%	11.276%	4.565%	76.099%	75.958%	89.071%	14.270%	80.542%
5	30.545%	29.780%	15.854%	55.682%	30.345%	3.480%	7.279%	76.460%	65.394%	37.799%	29.953%	63.212%	37.256%	72.727%
6	17.134%	10.509%	0.000%	1.842%	15.954%	52.813%	0.231%	98.126%	89.449%	12.902%	8.233%	67.794%	70.351%	60.000%
7	52.866%	71.016%	39.205%	60.848%	45.438%	23.133%	47.257%	57.562%	44.546%	43.801%	75.518%	77.464%	4.931%	50.226%
8	60.399%	49.963%	52.989%	48.887%	61.490%	18.329%	17.741%	43.870%	30.505%	74.142%	51.407%	74.267%	21.417%	79.121%
9	99.552%	89.116%	92.750%	56.843%	95.400%	12.969%	36.578%	4.639%	3.032%	65.920%	79.567%	90.477%	13.162%	82.353%
10	85.135%	90.161%	61.921%	50.696%	69.560%	12.255%	64.238%	33.490%	23.987%	48.162%	80.136%	91.710%	6.792%	75.758%
11	85.958%	71.247%	81.491%	57.831%	84.497%	11.444%	29.979%	17.036%	11.842%	70.978%	63.097%	83.901%	17.376%	73.190%
12	99.758%	95.126%	100.000%	59.293%	100.000%	7.201%	39.997%	0.000%	0.862%	70.733%	84.791%	98.751%	11.705%	85.560%
13	99.646%	97.142%	93.672%	57.698%	96.732%	7.589%	50.901%	4.049%	1.996%	70.005%	85.209%	96.475%	10.806%	86.536%
14	56.041%	38.785%	48.505%	55.551%	50.449%	44.702%	19.519%	48.869%	33.290%	93.702%	44.763%	77.422%	25.117%	71.875%
15	95.232%	98.923%	93.742%	60.761%	94.345%	3.222%	50.252%	10.926%	3.608%	85.303%	92.114%	97.950%	9.542%	86.421%
16	77.836%	77.445%	48.472%	62.132%	60.203%	17.622%	63.764%	46.802%	30.143%	57.905%	74.895%	86.889%	5.936%	47.119%
17	6.782%	12.344%	7.272%	55.263%	28.492%	100.000%	4.412%	98.412%	65.832%	4.299%	16.155%	49.829%	100.000%	60.000%
18	11.984%	14.816%	0.470%	100.000%	16.181%	32.589%	76.774%	94.114%	68.294%	3.444%	10.046%	33.019%	79.860%	42.857%
19	14.755%	31.664%	6.966%	71.228%	17.632%	52.813%	39.123%	87.571%	87.524%	52.712%	25.176%	90.306%	56.499%	46.667%
20	38.525%	54.945%	30.147%	70.452%	37.059%	29.633%	37.591%	70.533%	48.373%	59.307%	79.106%	63.553%	3.745%	34.503%
21	41.486%	35.837%	23.902%	64.946%	28.853%	42.528%	5.146%	67.257%	57.190%	53.902%	36.782%	75.158%	30.856%	51.282%
22	100.000%	90.494%	93.438%	57.568%	96.004%	10.040%	38.361%	4.750%	1.582%	64.212%	79.675%	91.958%	12.652%	88.070%
23	5.354%	59.807%	8.435%	51.012%	0.000%	63.702%	100.000%	90.809%	60.330%	52.674%	99.369%	54.178%	0.000%	7.692%
24	99.145%	96.175%	97.053%	62.948%	98.653%	4.508%	40.706%	2.850%	0.000%	80.737%	86.622%	96.800%	11.238%	91.781%
25	98.004%	99.590%	95.153%	58.319%	94.627%	9.196%	54.529%	7.378%	5.367%	66.303%	89.411%	100.000%	9.097%	83.575%
26	39.428%	15.728%	10.698%	98.804%	24.008%	35.653%	4.317%	69.156%	43.782%	49.858%	27.272%	68.073%	51.785%	45.455%
27	87.343%	94.945%	68.333%	54.873%	75.909%	3.526%	67.226%	23.009%	18.872%	43.401%	80.869%	89.951%	7.474%	73.729%
28	90.246%	79.358%	88.017%	56.584%	89.783%	24.838%	33.770%	10.425%	6.461%	62.806%	71.070%	90.166%	15.661%	82.278%
29	29.182%	14.439%	3.388%	70.000%	26.333%	52.813%	2.571%	89.082%	49.928%	67.359%	27.068%	67.298%	38.921%	80.000%
30	37.949%	30.861%	32.653%	67.105%	40.515%	46.633%	10.925%	63.137%	53.911%	68.972%	41.714%	78.951%	28.777%	69.048%
31	56.282%	44.677%	53.678%	55.263%	58.486%	39.421%	16.422%	43.118%	34.446%	86.347%	49.284%	80.755%	23.295%	54.054%
32	24.274%	16.455%	7.149%	37.579%	27.607%	62.250%	3.592%	86.255%	63.428%	50.846%	26.843%	51.410%	47.232%	52.000%
33	85.025%	76.933%	80.358%	60.479%	85.702%	16.940%	36.523%	16.157%	9.613%	65.439%	66.672%	87.306%	16.462%	77.551%
34	73.781%	62.792%	65.921%	50.834%	72.974%	26.013%	27.567%	32.391%	19.327%	67.492%	55.516%	75.579%	18.513%	75.610%
35	99.174%	89.332%	93.350%	63.070%	96.650%	10.461%	35.258%	5.150%	0.792%	66.622%	77.426%	93.900%	13.706%	81.440%
36	65.987%	60.535%	70.647%	53.070%	72.641%	28.095%	31.134%	31.784%	19.421%	95.483%	59.680%	78.795%	19.872%	63.810%
37	23.203%	14.828%	11.929%	23.026%	27.508%	11.523%	6.009%	82.107%	70.353%	6.751%	23.767%	20.942%	64.232%	0.000%
38	12.494%	9.782%	4.488%	61.404%	35.814%	100.000%	0.000%	100.000%	100.000%	0.000%	4.695%	70.521%	85.118%	100.000%
39	99.924%	90.887%	93.580%	57.782%	96.614%	14.956%	41.689%	5.195%	2.587%	67.570%	78.911%	94.412%	12.185%	79.887%
40	94.359%	100.000%	96.284%	60.519%	96.384%	5.489%	47.660%	6.865%	3.942%	83.315%	87.976%	91.613%	9.978%	84.150%
41	30.122%	49.836%	18.082%	56.598%	20.303%	33.322%	28.673%	79.730%	55.838%	53.369%	69.098%	66.023%	2.616%	14.493%
42	0.000%	0.000%	0.945%	0.000%	18.990%	100.000%	6.772%	97.689%	49.661%	40.599%	0.000%	0.000%	92.508%	100.000%
43	23.757%	23.228%	16.531%	66.062%	31.206%	59.321%	5.620%	76.518%	76.920%	26.562%	21.811%	75.373%	41.321%	24.138%
44	37.761%	32.996%	32.378%	64.654%	39.816%	49.112%	5.468%	65.098%	53.624%	100.000%	37.915%	75.982%	33.051%	68.627%
45	98.405%	97.616%	97.780%	60.562%	97.886%	7.564%	44.624%	5.673%	0.851%	85.918%	88.688%	96.967%	10.391%	84.110%
46	40.246%	30.532%	36.333%	64.654%	44.907%	72.243%	7.377%	59.301%	47.176%	67.778%	36.147%	73.611%	35.180%	49.020%
47	91.126%	85.442%	88.496%	57.856%	90.897%	0.000%	35.998%	11.084%	6.465%	68.656%	75.662%	90.620%	14.928%	81.457%
48	95.742%	97.046%	82.107%	58.640%	86.911%	2.799%	53.310%	17.852%	11.028%	61.613%	87.312%	90.325%	8.594%	77.132%
49	26.685%	54.921%	36.965%	53.753%	28.653%	45.850%	60.210%	60.472%	50.835%	79.118%	100.000%	82.023%	0.855%	11.475%

Findings - Clustering and Recommendation

Using kMeans in DataBricks, we clustered the titles into 50 clusters. We hoped to be able to do something like a recommendation based on this clustering. However, the results were often disappointing.

```
Choose a source title we are coming from:
```

source_details: pyspark.sql.dataframe.DataFrame = [track_id: string, artist_name: string ... 18 more fields]

	track_id	artist_name	track_name	cluster
1	3OBr2Y0n4S0BWwA7SxKfwU	Ludwig van Beethoven	Beethoven: 5 Variations on "Rule Britannia" in D Major, WoO 79: Theme. Tempo moderato	2

```
Source title belongs to cluster 2.

Top weights for that cluster are:
valence             0.874539
acousticness        0.832667
tempo               0.772181
instrumentalness    0.683628
liveness            0.603255
mode                0.518156
key                 0.507228
danceability        0.497327
speechiness         0.390702
time_signature      0.323944
```

```
Find source title in cluster, display +/- 5 rows:
```

index	track_id	artist_name	track_name
56	4WhVJNUCG4Sk3OC6ouXtNI	Ludwig van Beethoven	Beethoven: 24 Variations on Righini's Arietta "Venni amore" in D Major, WoO 65: Variation VI
57	4xx2sHYwBWhTXRGtCEdHtB	Johann Sebastian Bach	Partita No.1 In B-Flat Major, BWV 825: Menuet I da capo
58	1Ki55axMqIpWo9NWwfqC9c	Ludwig van Beethoven	Beethoven: 12 Variations on Haibel's "Menuet à la Viganò" in C Major, WoO 68: Variation II
59	1H5YA2K6z4d4xuSP4bV74a	Ludwig van Beethoven	Beethoven: 13 Variations on Dittersdorf's Arietta "Es war einmal ein alter Mann" in A Major, WoO 66: Variation IV
60	2PUsyTGrWLTLuMihu8iXcP	Ludwig van Beethoven	Beethoven: 12 Variations on a Russian Dance from Wranitzky's "The Forest Maiden" in A Major, WoO 71: Variation VIII
61	3OBr2Y0n4S0BWwA7SxKfwU	Ludwig van Beethoven	Beethoven: 5 Variations on "Rule Britannia" in D Major, WoO 79: Theme. Tempo moderato
62	6euqxexNlCOTDJXl6kgfDe	Ludwig van Beethoven	Beethoven: 33 Variations on a Waltz by Diabelli in C Major, Op. 120: Variation XXIII. Assai allegro
63	2Tks0FHE4KcztFPVmsuv2E	Ludwig van Beethoven	Beethoven: 5 Variations on "Rule Britannia" in D Major, WoO 79: Variation II
64	3ZEiEw1nSeEaIPRNMXKWAe	Ludwig van Beethoven	Beethoven: Piano Sonata No. 30 in E Major, Op. 109: III. (f) Variation V. Allegro, ma non troppo
65	2QWcqliGqe7sDVPoDYnUeK	Wolfgang Amadeus Mozart	Les petits riens, K.app.10 (ballet): Andantino
66	5Y0lzqNca12AfxOYzy4Sfm	Ludwig van Beethoven	Beethoven: 6 Variations on "Ich denke dein" in D Major, WoO 74: Variation II

Findings - Performance

Distributed-Computing vs Standalone

Identical kMeans-tasks were run in three different environments:

- AWS Sagemaker Cluster using an Sagemaker-internal Kmeans method
- Standalone execution within an AWS Sagemaker notebook using sklearn module
- Within a DataBricks workspace

The comparison of processing time was revealing:

Sagemaker Cluster-computing:

```
2020-12-03 21:11:14 Uploading - Uploading generated training model
2020-12-03 21:11:14 Completed - Training job completed
Training seconds: 46
Billable seconds: 46
CPU times: user 779 ms, sys: 34.6 ms, total: 813 ms
Wall time: 3min 12s
```

sklearn Standalone notebook: CPU times: user 2.65 s, sys: 228 ms, total: 2.88 s Wall time: 2.73 s

For the small data volume we worked on in this project, standalone sklearn was fastest by far. The overhead of starting up the Sagemaker cluster, or the distributed Hadoop clustering in Databricks was much too large for the execution time to matter at all.

Learning: Use distributed cluster-computing only if the data requires it. The time overhead is considerable.

Potential next steps

- Remembering strong bias to classical composers in the source data,
  a more balanced data source should be obtained.
- Within the existing data, a noticeable outlier-range of popularity close to or equal to zero was observed. Possibly, based on the other features, a prediction of popularity could improve the distribution of that feature and thus also improve clustering results.
- Use the text columns (artist, track_name) as features for the clustering,
  for example using word encoding
- Use clusters as target features in our dataset and train a separate classifier on them

DFKI project description:

Project Tasks - Status

The final goal is to find out new songs that can be part of your list of favourite songs. To accomplish this task, different ways of using a clustering approach are going to be compared. On the one hand, through a clustering approach following a distributed cluster-computing framework, and on the other hand, through the use of Amazon RDS. To do that, some tasks need to be accomplished:

1. Understand the content that is available in the dataset and clean the dataset if it is necessary.

2. Clustering following a distributed cluster-computing framework.
    a. Choose the best clustering algorithm taking into account the attributes that the dataset offers. In order to decide the best approach, it could be useful to have an overview of the different perspectives explaining the advantages and drawbacks of your decision.
    b. Evaluate your model statistically.

3. Clustering in the cloud using AWS.
    a. Create a database using Amazon RDS.
    b. Create tables to load the data of the dataset.
    c. Export the data into S3 and apply a cluster analysis in AWS using a different clustering algorithm.
    d. Analyze the outcomes.
    e. (Optional task) Put the final results into RDS for future access.

4. Use the different packages of visualization explained during the course to visualize clusters based on your findings.

5. Compare the findings of both methods using metrics, the visualizations, etc.

Dataset

5

Spotify Data With Audio Features

Datasets with audio features for over 20k songs, retrieved from Spotify.

Rafael Duarte • updated 9 months ago (Version 1)

MISSION: ACCOMPLISHED

Thank you for your interest