Clustering of Songs with Amazon Web Services

Certification project
for AIDA 2020

Falk Lutz
Julian Godley
Simon Harston

Table of Contents

Table of Contents 2

List of Abbreviations 2

Assignment 3

Prerequisites 3

- Technical infrastructure 3
- Data availability 3

Process workflow / pipeline 4

- Data preprocessing 4
- Exploratory Data Analysis (EDA) 4
- Clustering Algorithms 7
- Distributed Cluster-Computing Frameworks 7
  - AWS Cloud 7
    - AWS Sagemaker Standalone Notebook 7
    - AWS Sagemaker Cluster 9
  - Apache Spark on Databricks 9

Learnings / Insights 10

- Clustering and Recommendation 10
- Performance 10

Potential next steps 11

Distribution of tasks 11

List of links 11

- Kaggle Challenge site 11
- Spotify 11
- Clustering Algorithms 12
  - Kmeans 12
  - DBScan 12

List of Abbreviations

AWS	Amazon Web Services
CSV	“Comma separated values”, a format where data columns are separated by comma
DBSCAN	“Density-based spatial clustering of applications with noise”, a clustering method
EDA	Exploratory Data Analysis
PCA	Principal Components Analysis, a method of reducing feature dimensions
RDS	Relational Database Service (part of Amazon Web Services)
S3	Simple Storage Service (part Amazon of Amazon Web Services)

Assignment

The objective of this project is to group a set of songs allocated in Spotify that have some similarities among them. This dataset is composed of three files retrieved from Spotify with audio features for over 20k songs. Every file contains 18 attributes among which are strings and integer attributes such as artist name, track id, danceability, energy, etc.

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.

Prerequisites

Technical infrastructure

The project team agreed to utilize the same technical infrastructure as had been utilized for a previous project. This included:

- Data Storage on an AWS S3 Bucket (s3://flutz-bucket/spotify/)
- Database on AWS RDS (fjs-project.cpfawmiirogo.eu-central-1.rds.amazonaws.com)
- Coding platforms
    - AWS Sagemaker Notebook Instance (https://flutz.notebook.eu-central-1.sagemaker.aws/tree/CoS_AWS)
    - Databricks workspace (https://community.cloud.databricks.com/?o=6620618385044013)
    - Repository: GitHub (https://github.com/jgo-DA/CoS_AWS)

Data availability

To ensure availability to the whole project team, the data files were downloaded from the kaggle website stated in the project brief and stored on the project storage location at Amazon S3:

- 2020_spotify.csv (252.31 KB)
- brazil_data.csv (1.29 MB)
- world_data.csv (1.28 MB)

Process workflow / pipeline

The project team agreed on the following workflow to conduct the required tasks.

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 CSV-files 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).

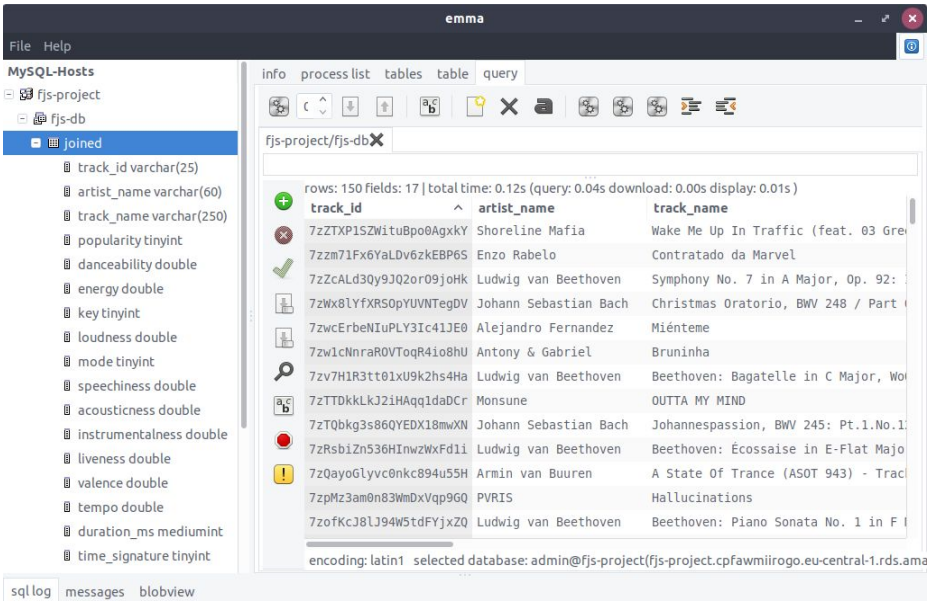
```
# Now some more complicated duplicates - same artist, track_name and duration

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	instrumen
4268	As I Lay Dying	Blinded	5xa5C8TmCuF1q8cBRutiMY	62	0.339	0.9720	8	-5.260	1	0.1850	0.000006	0.
8774	As I Lay Dying	Blinded	2HdjEa5BP2VACt1velDTIk	56	0.332	0.9720	8	-5.258	1	0.1980	0.000006	0.
3911	As I Lay Dying	My Own Grave	6bDoeNLCA32SYhTzIk5w5y	61	0.475	0.9940	5	-4.978	0	0.2500	0.000113	0.
6751	As I Lay Dying	My Own Grave	0CcqWuAEJC93K8cBMbAjgI	57	0.477	0.9940	5	-4.953	0	0.2410	0.000115	0.
3110	Bastille	Admit Defeat	1OHAFggB1Jatvto7HvUN4L	56	0.653	0.6410	5	-7.007	1	0.0475	0.155000	0.
1952	Bastille	Admit Defeat	4qcnL8k4i8Ynw7kAPgVoD6	54	0.651	0.6400	5	-7.019	1	0.0460	0.149000	0.
6764	Cigarettes After Sex	Cry	0r0zdaQ9S3fouDnvJ25pwI	63	0.408	0.3970	7	-10.452	1	0.0279	0.756000	0.
8581	Cigarettes After Sex	Cry	0Qr61NXlyAeQaADO5xn3rI	53	0.409	0.3990	7	-10.456	1	0.0275	0.763000	0.

- The joined dataset was written to a CSV-file on our S3 bucket
  and to a table on an RDS database instance.



Exploratory Data Analysis (EDA)

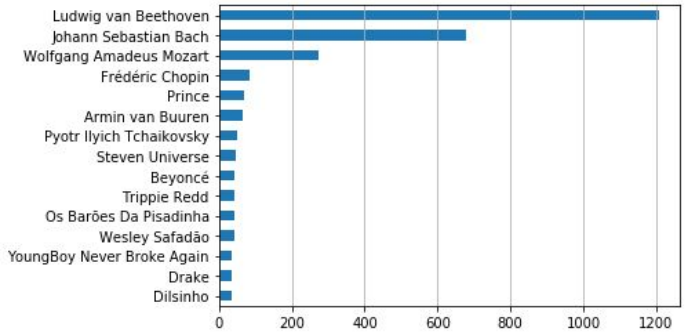
- Analysis of the data completeness showed that in all 14 feature columns,
  there were no null values.

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

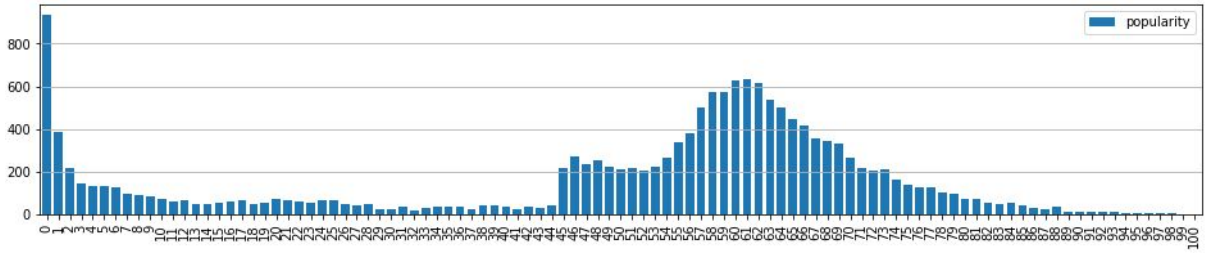
column	count	mean	std	min	25%	50%	75%	max
popularity	15626	49.098234	24.273116	0.00000	43.00000	58.000000	65.000000	100.000
danceability	15626	0.594562	0.195956	0.00000	0.45800	0.622000	0.745000	0.983
energy	15626	0.534537	0.271949	0.00002	0.33700	0.588000	0.750000	1.000
key	15626	5.183348	3.594108	0.00000	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

- Analysis of the “artist” and “popularity” features showed a very strong bias to classical music: of 15,626 rows, 1206 alone belong to Ludwig van Beethoven, a further 679 to Johann Sebastian Bach and another 275 to Wolfgang Amadeus Mozart.
  These three composers already cover roughly 14% of all titles!

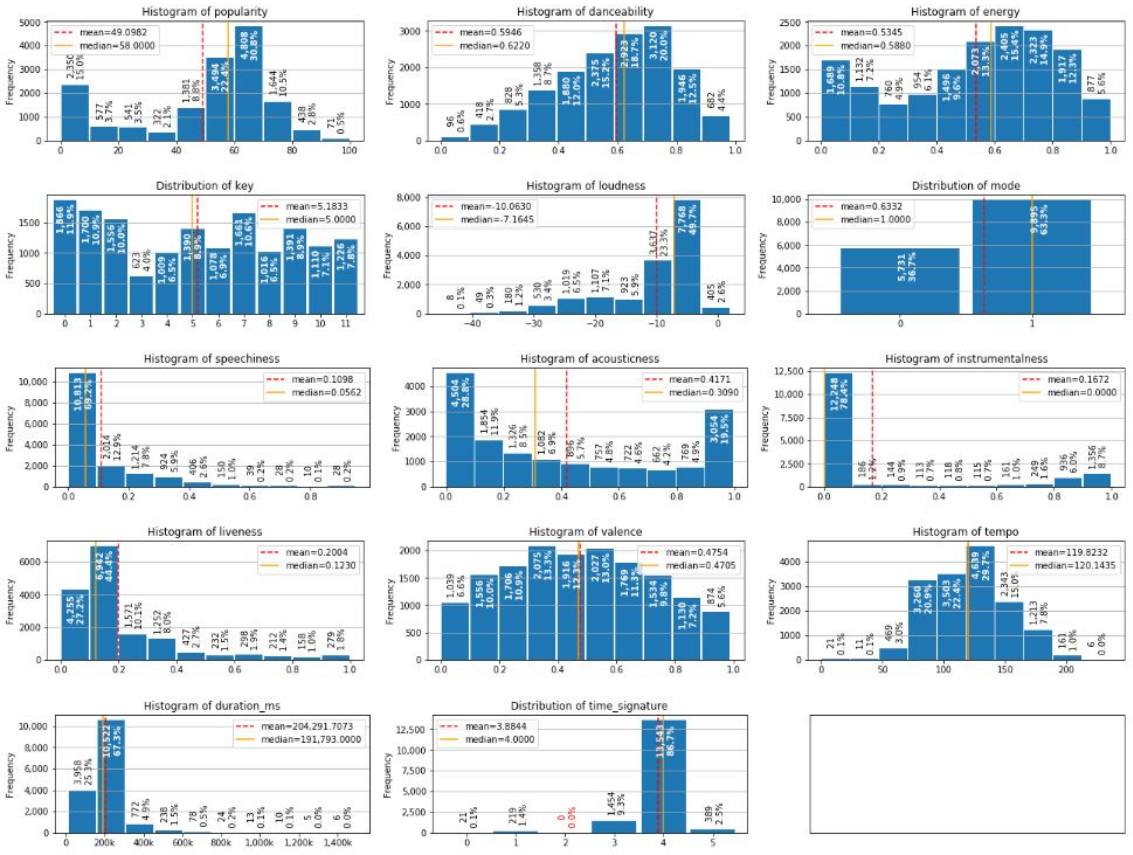
```
df_joined['artist_name'].value_counts()[15::-1].plot.barh().grid(axis='x')
```



- 938 rows (6%) of all titles have a popularity of zero, which seems suspect.



- 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.

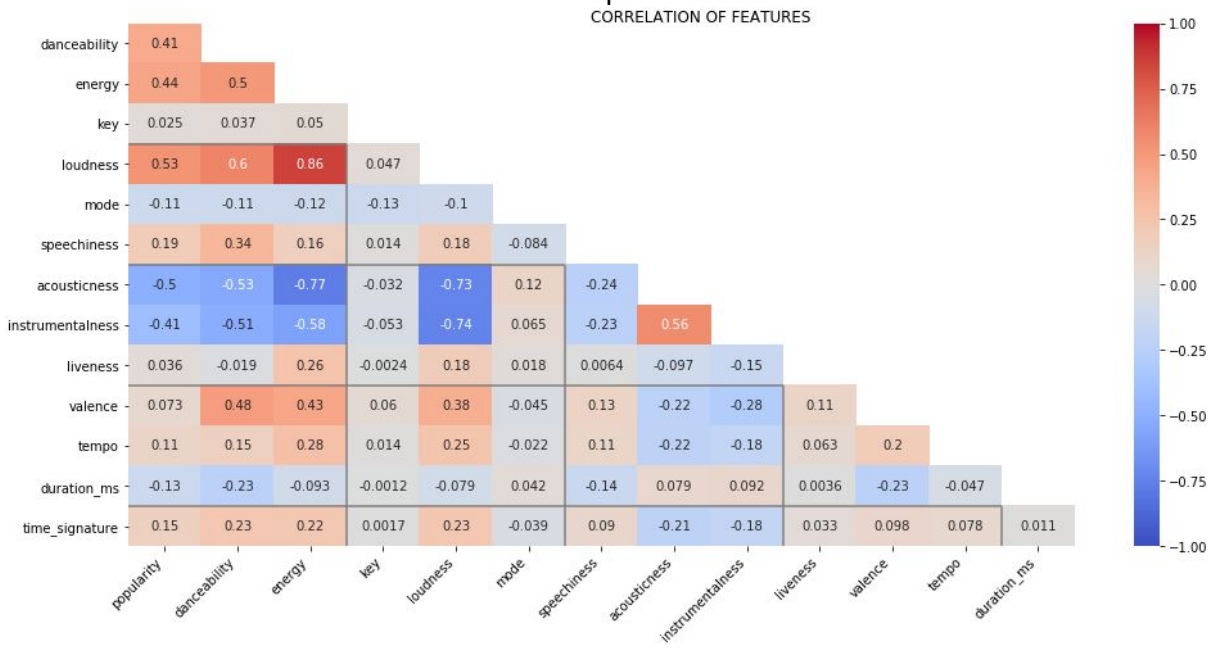
```
print(f'Total rows of data in df_joined dataframe: {len(df_joined):,}')

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
```

- Some playful comparisons of titles from different artists were performed.
- Feature correlation was visualized in a heatmap:



Clustering Algorithms

The task of clustering a dataset without a target value (label) is leading to the approach of unsupervised learning. With clustering there is actually no right or wrong in the results but it is about finding a latent structure within the data. The word ‘latent’ comes even more into play when looking at the data. We don’t want the songs to be categorized in the standard way (e.g. ‘Pop’, ‘Rock’ etc. or by artist), because this would rather be a question of filtering, but to find similarities based on song features like ‘valence’, ‘instrumentalness’ etc.

For this, the most commonly used clustering algorithms were applied here:

- K-means
- Hierarchical
- DBScan
  The DBScan algorithm is recommended, when the K-means and hierarchical reach their limits due to having to handle noise and generating bad results with finding clusters of nonspherical shape. (see Amit Shreiber, A Practical Guide to DBSCAN Method)

Distributed Cluster-Computing Frameworks

AWS Cloud

AWS Sagemaker Standalone Notebook

Initial Validation using Kmeans model from sklearn
To enable a direct comparison between the distributed and standalone computing frameworks the team chose to implement the primary clustering model also on an AWS Sagemaker Standalone Notebook. The implementation (K-Means-SKLearn_jgo.ipynb) required a standard Jupyter notebook and utilized a sklearn kmeans model. The algorithm was set with a parameter of n_clusters=24 (the same setting as on the distributed-computing experiment).
The notebook completed the computation with the following processing times captured:

    user 2.65 s, sys: 228 ms, total: 2.88 s Wall time: 2.73

This shows that the project dataset does not require large computational resources.
Therefore it was decided to implement other models on the standalone notebook setup.

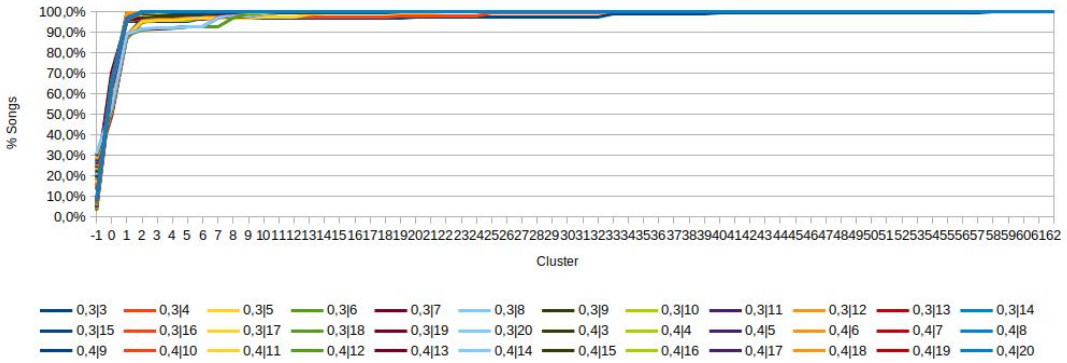
DBScan model from sklearn
DBScan was selected as the second clustering model and for this assessment the sklearn model was implemented. The initial implementation (DBScan_jgo.ipynb) utilized the full dataset (joined.csv) and included basic EDA (heatmap of feature correlation) and data preprocessing (dropping of text based columns and scaling of numeric values) steps for transparency (the detailed process was being developed at this time).

Before the data could be clustered it was necessary to assess which hyperparameter set (eps and min_samples) would best suit the task.
This was conducted by using a nested for-loop with ranges of eps and min_samples values.
Unfortunately the activity led to frequent outages due to Gateway timeout and dead kernel errors between the Jupyter Notebook and the AWS Sagemaker server backend.

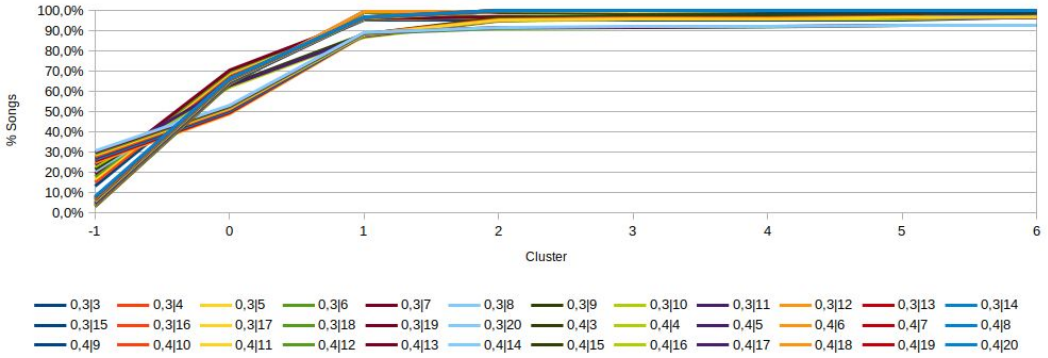
Reassessment of the source data during this phase gave rise to the insight that the zero popularity values were an anomaly and should be removed to provide a “train” dataset and a potential “test“ dataset for future prediction efforts should they become relevant.

In a second iteration of the notebook the hyperparameter assessment was split into multiple cells to reduce individual processing time of individual cells and to avoid a timeout error that occurs after a cell “walltime” of approximately 10 minutes.

This modification resulted in completed hyperparameter assessment and the resultant cluster dynamics were then transferred to a diagram showing the cumulative percentage of total songs for the relevant clusters by hyperparameter set (shown in the legend as eps|min_samples) for evaluation.



Here the same diagram but zoomed into the left hand section to show the relevant section.



Following guidance on selecting parameters for DBScan:
"... According to a research made in 2017 by Schubert, Sander, et al, the desirable amount of noise will usually be between 1% and 30%.
Another insight from that research is that if one of the clusters contains many (20%-50%) points of the dataset, it indicates that you should choose a smaller value for ε or to try another clustering method. ..."
(Amit Shreiber, A Practical Guide to DBSCAN Method)

Two sets were selected: eps=0.3, min_samples=5 and eps=0.3, min_samples=18

The first was deployed in a separate notebook (DBScan_deployment_jgo.ipynb) which generated a first clustering.

The notebook was modified to take the insight on excluding zero “popularity” values. This was performed in the data preprocessing section. This notebook is named DBScan_no_pop0_jgo.ipynb.
The results were more meaningful and as a final modification toward a faster and less error prone version, the individual pipeline process modules were split into the following separate notebooks:

Data Pre-Processing (Data_Wrangle_no_pop0_jgo.ipynb)
Steps include:

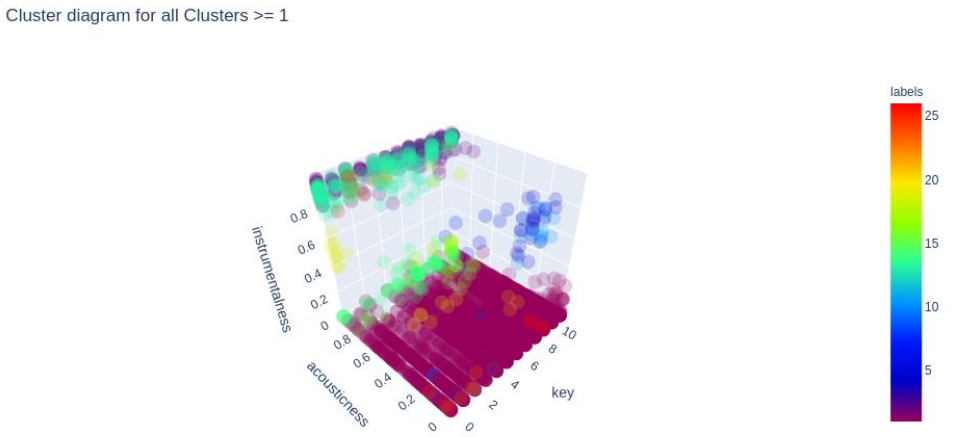
- loading source data from RDS (15626 rows × 17 columns)
- Split into:
  df_train: (14688, 17)
  df_test : (938, 17)

Encoding of “train” data (Encoding_Train_dataset_no_pop0_jgo.ipynb)
Steps include:

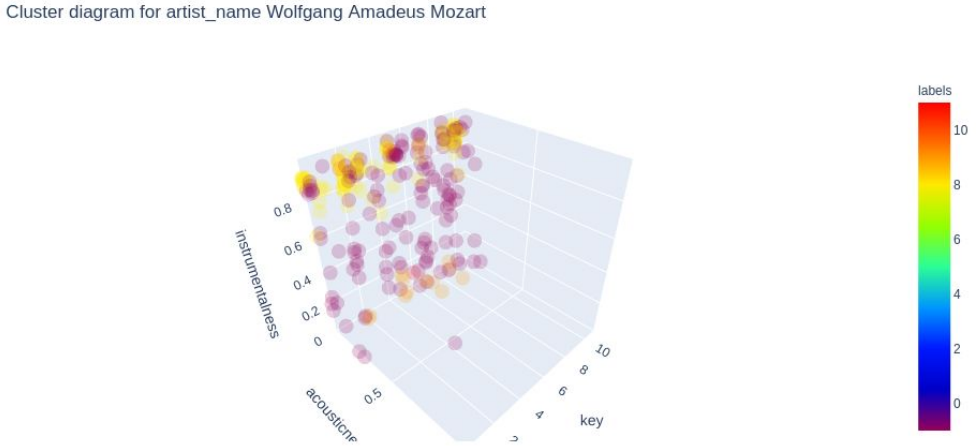
- dropping non numeric columns (‘artist_name’ and ‘track_name’
- scaling of numeric columns

Deployment of selected hyperparameter
sets 0.3 | 5eps=0.3, min_samples=5 and 0.3 | 18 eps=0.3, min_samples=18
(DBScan_deployment_no_pop0_jgo.ipynb and DBScan_deployment_no_pop0_jgo-set2.ipynb)
Steps include:

- Clustering to resulting clusters
- Visualization of Clusters and individual artists using 3d scatter plots in Plotly
  e.g. Clusters >= 0



e.g. WAM Tracks



- Storing resultant DataFrame to RDS

Clustering using the Amazon SageMaker Data science workflow

Even though the procedure of applying the various machine learning algorithms is generally following the same steps, for this task there are several requirements we had to meet:

- ★ Training and Deployment jobs run on scalable AWS instances
- ★ k-means clustering using built-in Amazon SageMaker algorithms
- ★ Data and artifact storage in AWS S3 bucket

Therefore the following steps were proceeded (Jupyter notebook *kmeans-sagemaker-flutz*):

1. Loading cleaned data from RDS instance
2. Preprocessing: Scaling data

Since k-means is based on the relative distances between different features it is almost mandatory to transform numerical columns to a range between 0 and 1

3. Preprocessing: PCA

Generally PCA is a feature reduction algorithm. Given 14 features, the question was, if a feature reduction would make sense here. We decided to make use of PCA here for the following reasons:

a) Even though this dataset is rather tiny, in a big data context it would make sense to keep the feature set as small as possible

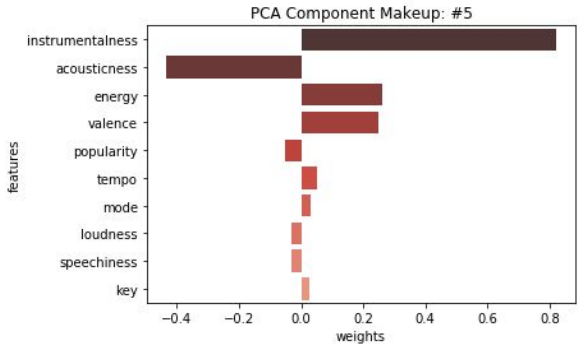
b) PCA also serves for data compression and noise reduction which could be useful

The drawback of PCA is that it becomes difficult to interpret the results of the clustering (since they do not base on the original features).

Since PCA is an unsupervised machine learning algorithm of its own, the following steps had to be performed:

a) Model training and fitting on AWS ML instance:
   - Storing model artifact on Amazon S3.
b) Accessing the PCA model attributes:
   - loading model from S3 - uncompressing - converting from MXNet format
c) Calculating explained-variance-ratio and determining number of components

d) Interpreting components by weightings of the original features and giving them descriptive names e.g. 'instrumental_positive_energy'



e) Model deployment on AWS instance

4. K-means clustering

a) Model training and fitting on AWS ML instance:

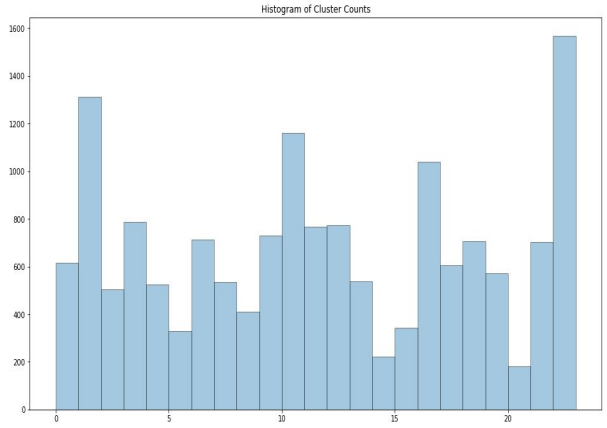
- in this case we were setting the numbers of clusters intentionally to 24 (no elbow method) since we wanted a rather high number of clusters in order to make the more significant for possible recommendations - futile, anyway.
- Storing model artifact on Amazon S3.

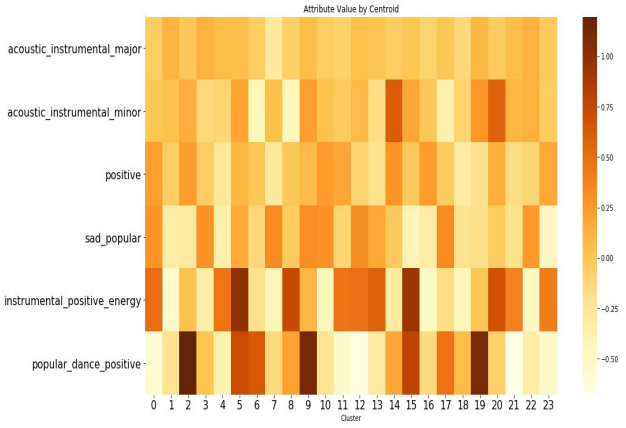
b) Model deployment on AWS instance

c) Accessing the PCA model attributes:

- loading model from S3 - uncompressing - converting from MXNet format

5. Analyzing clusters and trying to interpret their composition





6. Cluster prediction

- making a cluster prediction for a song taken out of the dataset beforehand

Clustering using a distributed cluster-computing framework

As mentioned before, the general workflow is similar to the one before but the different requirements had to be met here as well:

- ★ 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

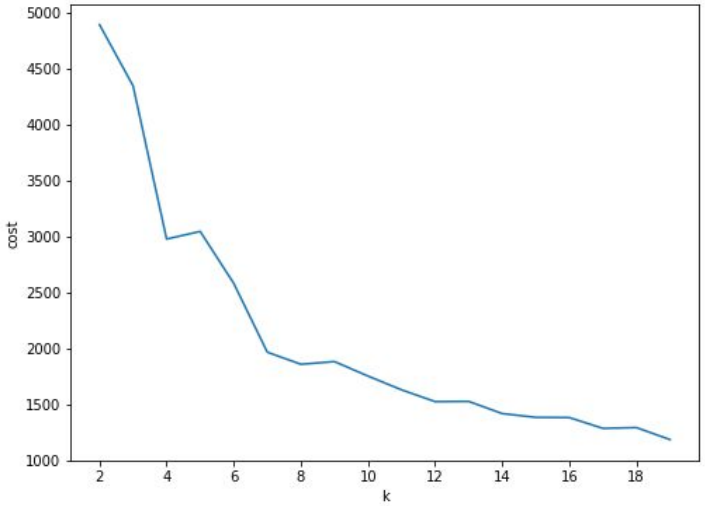
Taking this into account the workflow was as follows (Jupyter notebook *kmeans-pyspark*):

1. Loading cleaned data from RDS instance
2. Preprocessing: Scaling data
3. Preprocessing: Vectorize feature columns

- Spark ML requirement

4. Identifying the optimal number of clusters

- Calculating the euclidean distance of cluster centroids for various k
- “Elbow effect”: The point where the decrease sharply diminishes should be the optimal k. In our case: 7



5. K-means model training and data transforming
6. Cluster interpretation

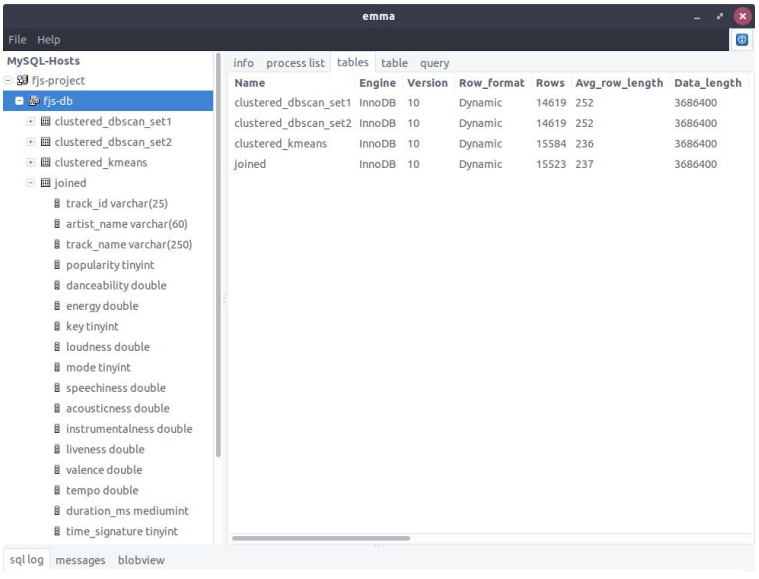
cluster	popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
0	91.060%	93.953%	78.392%	60.259%	81.424%	9.524%	55.091%	19.930%	16.805%	49.650%	86.740%	99.555%	8.072%	67.373%
1	23.933%	19.662%	15.867%	68.660%	29.358%	14.205%	3.636%	79.042%	67.330%	47.840%	23.913%	78.842%	43.947%	54.545%
2	22.828%	49.733%	15.214%	50.723%	40.170%	51.816%	39.070%	83.287%	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%	89.110%	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	95.233%	98.923%	93.742%	60.761%	94.345%	3.722%	50.252%	10.825%	3.600%	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%	5.145%	67.257%	57.190%	53.902%	36.782%	75.159%	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%	91.953%	12.652%	88.970%
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%	80.889%	89.951%	7.474%	73.729%
28	90.245%	79.358%	88.017%	56.594%	89.783%	24.838%	33.770%	10.425%	8.461%	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%	99.387%	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%	5.480%	47.660%	0.851%	3.942%	83.315%	87.978%	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%	7.977%	59.301%	47.176%	67.778%	36.147%	73.611%	35.180%	49.020%
47	91.126%	85.442%	88.498%	57.856%	90.897%	0.000%	35.998%	11.084%	6.465%	68.656%	75.652%	90.620%	14.929%	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%

7. Cluster predictions
8. Evaluating predictions

- Calculating silhouette score

9. Making a recommendations based on cluster prediction
10. Writing data to RDS instance

- Clustering assignment was written back to the RDS instance



Learnings / Insights

Clustering and Recommendation

- From the project description, we understood the primary goal to be clustering of the titles, which we performed successfully (albeit the available features produced sometimes “weird” cluster companions).
- As a secondary goal, we understood a kind of recommendation based on the clustering should be evaluated. This target can fairly squarely be regarded as failed, because the “weird” cluster composition mentioned above makes good recommendations extremely unlikely.
  An example: we found no less than 36 titles belong to one recording of the opera “Leonore”.
  Due to large variations in feature assignment (in an opera, you have arias, a-cappellas, choruses, interludes, you have many different tempi and pieces in various keys, happy and sad moods...), these 36 titles were clustered by kMeans into no less than 23 different clusters!
  Obviously, a “recommendation” for even the next piece within the opera based on this cluster assignment failed miserably. Instead, probable outliers in the clustering joined pieces from Ludwig van Beethoven and Kanye West in the same cluster!

Performance

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:

```
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
```

Sagemaker Cluster-computing:

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

Distribution of tasks

The individuals project members executed the following project tasks:

- Falk Lutz
  - kMeans clustering in SageMaker and DataBricks
- Julian Godley
  - Source data and infrastructure assurance
  - Clustering methods on Standalone Notebooks: DBScan, kMeans
- Simon Harston
  - CSV-file loading, concatenation, data preprocessing, exploratory data analysis
  - storage of source data and clustered data to S3 bucket and RDS database
  - setup of DataBricks environment and loading of data from RDS instance
  - debugging of kMeans clustering on DataBricks including
    - heatmap-style display of feature-to-cluster weights
    - thoughts on “recommendations” based on cluster assignment

List of links

Kaggle Challenge and other websites dealing with the idea

- Challenge site
  https://www.kaggle.com/rafaelnduarte/spotify-data-with-audio-features
- “Clustering the Most Listened to Songs of the 2010s Using Spotify Data.”
  https://medium.com/analytics-vidhya/clustering-most-listened-songs-of-the-2010s-using-spotify-data-8e25e8b082ce
- Discovering similarities across my Spotify music using data, clustering and visualization
  https://towardsdatascience.com/discovering-similarities-across-my-spotify-music-using-data-clustering-and-visualization-52b58e6f547b

Spotify

- Glossary
  https://artists.spotify.com/blog/glossary-of-music-terms-actual-music-terms
- Documentation on audio analysis
  https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-analysis/
- Insight on Valence
  https://community.spotify.com/t5/Content-Questions/Valence-as-a-measure-of-happiness/td-p/4385221
- Insight on Mode & key
  https://developer.spotify.com/documentation/web-api/reference/tracks/get-several-audio-features/

Clustering Algorithms

Kmeans

Walker Rowe, Amazon SageMaker: A Hands-On Introduction
https://www.bmc.com/blogs/amazon-sagemaker/

DBScan

Amit Shreiber, A Practical Guide to DBSCAN Method,
https://towardsdatascience.com/a-practical-guide-to-dbscan-method-d4ec5ab2bc99

Other

- Documentation Pandas module
  https://pandas.pydata.org/pandas-docs/stable/reference/index.html
- Documentation PySpark SQL module
  https://spark.apache.org/docs/latest/api/python/pyspark.sql.html
- Documentation PySpark Machine learning module
  http://spark.apache.org/docs/latest/api/python/pyspark.ml.html