

BDA, Praktikumsbericht 1

Gruppe mi6xc: Alexander Kniesz, Maximilian Neudert, Oskar Rudolf

Aufgabe 1

Zuerst haben wir ein gemeinsames Notebook [bericht1](#) auf Zeppelin mit Ownern aller Gruppenmitgliedern erstellt, auf dem wir gemeinsam arbeiten können.

Wir haben als ApplicationID `app-20190425183601-0341` erhalten und uns auf `http://141.100.62.85:8080/` die Ressourcen angeschaut. Auffällig war, dass keine Kerne zugewiesen waren. Wenn man Testweise eine Endlosschleife mit PySpark ausgeführt hat, dann ging der Status auf Waiting. Wir sind von dem Monitor noch nicht ganz überzeugt. Der Status wirkt ziemlich träge. Aber man kann damit gut Applications abschießen die in Jobs festhängen.

Zuerst haben wir uns alle Million Song relevanten Tabellen ausgegeben lassen:

```
%pyspark
spark.sql("show tables like 'msd10k*']").show(truncate=False)
```

```
+-----+-----+-----+
|database|tableName          |isTemporary|
+-----+-----+-----+
|default |msd10k_more_metadata|false      |
|default |msd10k_more_metadata_row|false     |
|default |msd10k_some_metadata |false      |
|default |msd10k_some_metadata_row|false     |
|default |msd10k_timbre        |false      |
|default |msd10k_timbre_row    |false      |
+-----+-----+-----+
```

Dann haben wir die Daten gesichtet:

```
%sql
select * from msd10k_timbre limit 100
select * from msd10k_some_metadata limit 100
select * from msd10k_more_metadata limit 100
```

title ▾	track_id ▾	timbre_0 ▾	timbre_1 ▾	timbre_2 ▾	≡
Doppelgänger [Qliphothic Phantasmagoria]	TRABEFN128F92D92 5B	21.613	-136.784	-105.838	
Doppelgänger [Qliphothic Phantasmagoria]	TRABEFN128F92D92 5B	21.864	-151.802	-100.926	
Doppelgänger [Qliphothic Phantasmagoria]	TRABEFN128F92D92 5B	20.972	-156.087	-94.076	
Doppelgänger [Qliphothic Phantasmagoria]	TRABEFN128F92D92 5B	22.255	-145.706	-81.169	

Wir haben vorerst geprüft, ob die Timbre überall gleich lang sind

```
%pyspark
s1 = 'TRAVHPV128F933E986'
s2 = 'TRAKXYJ128F42525ED'
def get_tdur(track_id):
    not_sql_df = spark.sql("select count(timbre_0) as val from
msd10k_timbre where track_id = '{}'.format(track_id))
    s_tcount = not_sql_df.collect()[0]['val']
    not_sql_df = spark.sql("select duration as dur from
msd10k_more_metadata where track_id = '{}'.format(track_id))
    s_duration = not_sql_df.collect()[0]['dur']
    timbre_duration = s_duration / s_tcount
    return timbre_duration

d1 = get_tdur(s1)
d2 = get_tdur(s2)

print(d2 - d1)
```

Wir haben **0.0299464126059322** als Ergebnis bekommen, was bedeutet, dass die Timbre nicht gleich lang sind.

Beispielhaft lassen wir uns für **duration** und **loudness** eine statistische Zusammenfassung mittels **describe()** geben:

```
%pyspark
df = spark.sql("select duration, loudness from msd10k_some_metadata")
df.describe().show()
```

```
+-----+-----+-----+
|summary|      duration|      loudness|
+-----+-----+-----+
|  count|      10000|      10000|
|   mean|238.50751842799997|-10.485668499999996|
|  stddev|114.13751356561322|   5.39978822917156|
|   min|       1.04444|      -51.643|
|   max|     1819.76771|       0.566|
+-----+-----+-----+
```

Beim Vergleich der Performance haben wir durch Sichtprüfung mehrerer runs einmal mit `describe` einmal mit Aggregationsfunktionen column based und row based verglichen und kamen zum Ergebnis, dass row based langsamer läuft.

```
%pyspark
from pyspark.sql import functions as F
df = spark.sql("select duration from msd1m_some_metadata")
df.describe().show()
```

```
+-----+-----+
|summary|      duration|
+-----+-----+
|  count|      1000053|
|   mean|249.5008840431487|
| stddev|126.2293871957265|
|    min|         0.31302|
|    max|      3034.90567|
+-----+-----+
```

Took 1 sec. Last updated by istmnneud at April 25 2019, 7:04:38 PM.

```
%pyspark
from pyspark.sql import functions as F
df = spark.sql("select duration from msd1m_some_metadata_row")
df.describe().show()
```

```
+-----+-----+
|summary|      duration|
+-----+-----+
|  count|      1000053|
|   mean|249.5008840431487|
| stddev|126.2293871957265|
|    min|         0.31302|
|    max|      3034.90567|
+-----+-----+
```

Took 4 sec. Last updated by istmnneud at April 25 2019, 7:04:33 PM.

```
%pyspark
from pyspark.sql import functions as F
df = spark.sql("select duration from msd1m_some_metadata")
df.agg(F.min(df.duration),F.max(df.duration),F.avg(df.duration),F.sum(df.duration)).show()
```

```
+-----+-----+-----+-----+
|min(duration)|max(duration)|  avg(duration)|  sum(duration)|
+-----+-----+-----+-----+
|      0.31302|    3034.90567|249.5008840431487|2.4951410759000298E8|
+-----+-----+-----+-----+
```

Took 1 sec. Last updated by istmnneud at April 25 2019, 7:01:59 PM.

```
%pyspark
from pyspark.sql import functions as F
df = spark.sql("select duration from msd1m_some_metadata_row")
df.agg(F.min(df.duration),F.max(df.duration),F.avg(df.duration),F.sum(df.duration)).show()
```

```
+-----+-----+-----+-----+
|min(duration)|max(duration)|  avg(duration)|  sum(duration)|
```

min(qdr:accn),max(qdr:accn),avg(qdr:accn),sum(qdr:accn),

+-----+-----+-----+-----+

|0.31302|3034.90567|249.5008840431487|2.4951410759000298E8|

+-----+-----+-----+-----+

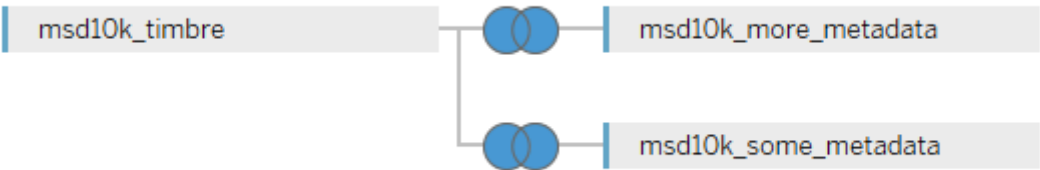
Took 5 sec

Last updated by istmnneud at April 25 2019, 7:01:56 PM.





Aufgabe 2

a)

Wir haben die Joins nach folgendem Schema durchgeführt:



Beide Joins wurden über die Track Id durchgeführt:

Join			
 Inner	 Left	 Right	 Full Outer
Data Source		msd10k_more_metad...	
Track Id	=	Track Id (Msd10K ...	
Add new join clause			

b)

Skalierung:

Die Daten werden von Tableau in zwei Kategorien eingeteilt: Dimensionen und Maßzahlen. Innerhalb der Variablen der Dimensions-Kategorie sind die kategorialen Merkmale (qualitativen), nach denen sich z.B. gut aggregieren lässt. Bei den Maßzahlen handelt es sich um metrisch Skalierte (quantitative) Variablen.

Missing Values

Obwohl es in Tableau möglich ist, sich fehlende Werte anzeigen zu lassen (z.B. über die folgende Darstellung), handelt es sich bei dem Tool eher um ein Visualisierungstool und die Analyse von Missings müsste für jede Variable einzeln mittels Grafik durchgeführt werden.



Für eine schnellere Analyse der Missing-Data haben wir uns mittels R einen schnellen Überblick verschafft:

```
require(data.table)

missing_names <- c("Variable","NA_count","empty_string_count", "0_count" )

setwd("C:\\Users\\rudol\\Documents\\AAA_Wichtig\\STUDIUM\\MSc. Data
Science\\2. Semester\\Big_Data_Analytics\\Datasets")

# Anzahl Missing Values im TimbreDatensatz:

timbres <- as.data.frame(fread("msd10k_timbre.tsv"))

timbre_missings <- data.frame (names(timbres))

timbre_missings <- cbind(timbre_missings,apply(timbres, function(x)
sum(is.na(x))))
timbre_missings <- cbind(timbre_missings,apply(timbres, function(x)
sum(x=="")))
timbre_missings <- cbind(timbre_missings,apply(timbres, function(x)
sum(x==0)))

# Spaltennamen
names(timbre_missings) <- missing_names

# Anzahl Missing Values im MetaDatensatz (some + more enthalten viele
Redundanzen, daher hier nur "more"):

meta_data <- as.data.frame(fread("msd10k_more_metadata.tsv"))

# Anzahl Missing Values im TimbreDatensatz:

meta_data_missings <- data.frame (names(meta_data))

meta_data_missings <- cbind(meta_data_missings,apply(meta_data,
function(x) sum(is.na(x))))
meta_data_missings <- cbind(meta_data_missings,apply(meta_data,
function(x) sum(x=="")))
meta_data_missings <- cbind(meta_data_missings,apply(meta_data,
function(x) sum(x==0)))

# Spaltennamen
names(meta_data_missings) <- missing_names
```

Hier ist das Ergebnis abgebildet:

Variable	NA_count	empty_string_count	0_count
V1	0	822	0
V2	0	0	0
V3	0	0	5010
V4	0	0	51
V5	0	0	64
V6	0	0	106
V7	0	0	105
V8	0	0	91
V9	0	0	112
V10	0	0	172
V11	0	0	167
V12	0	0	197
V13	0	0	255
V14	0	0	178

Timbre_Daten:

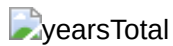
Variable	NA_count	empty_string_count	0_count
V1	0	0	0
V2	0	1	0
V3	0	0	0
V4	0	0	0
V5	0	0	25
V6	0	0	5320
V7	6258	0	NA
V8	6258	0	NA
V9	4	0	NA
V10	4352	0	NA
V11	0	0	0
V12	0	0	0
V13	0	0	0
V14	0	0	1213
V15	0	0	523
V16	0	0	3089
V17	0	0	277
V18	0	0	3
V19	0	0	2167

Meta_Daten:

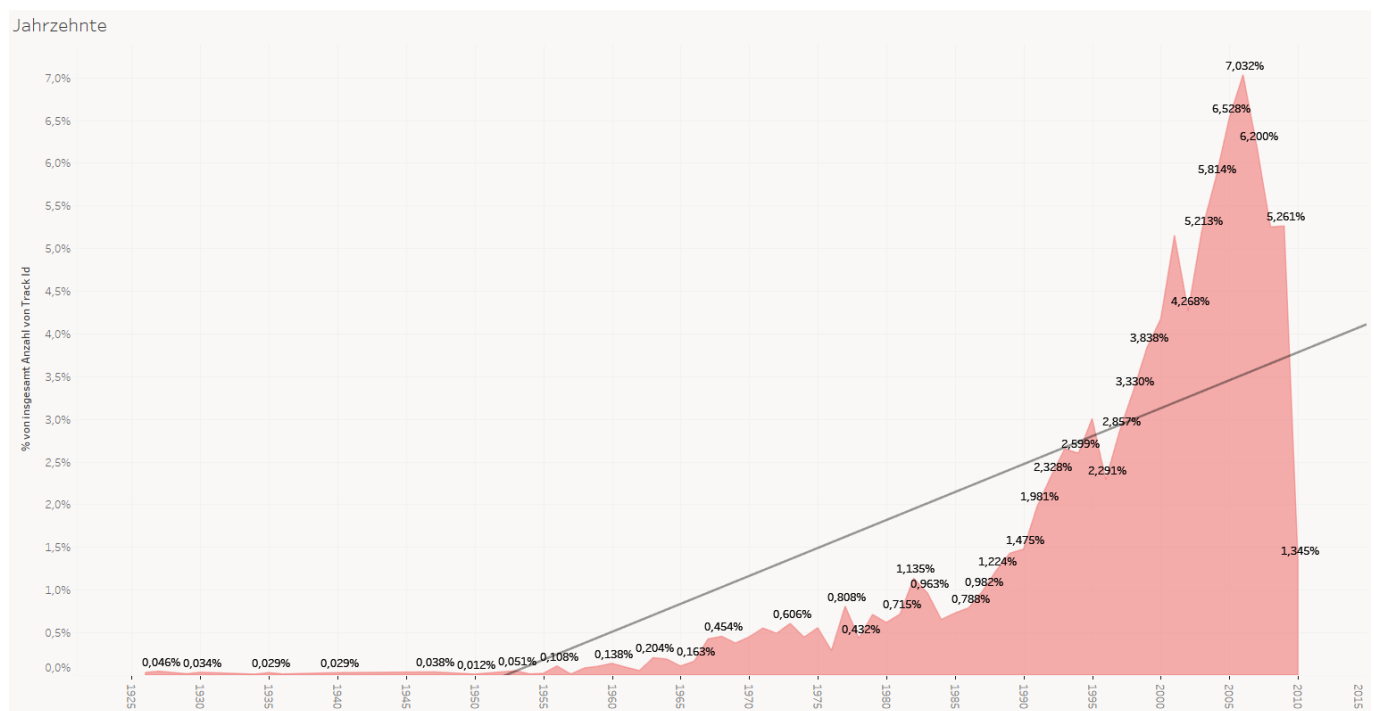
Diagramm mit Jahreszahlen

Nach herausfiltern der "überflüssigen" Nullwerte (Jahr==0) sind wir auf folgende Übersicht über die Jahrzente gekommen:

Insgesamt:



In Prozent:



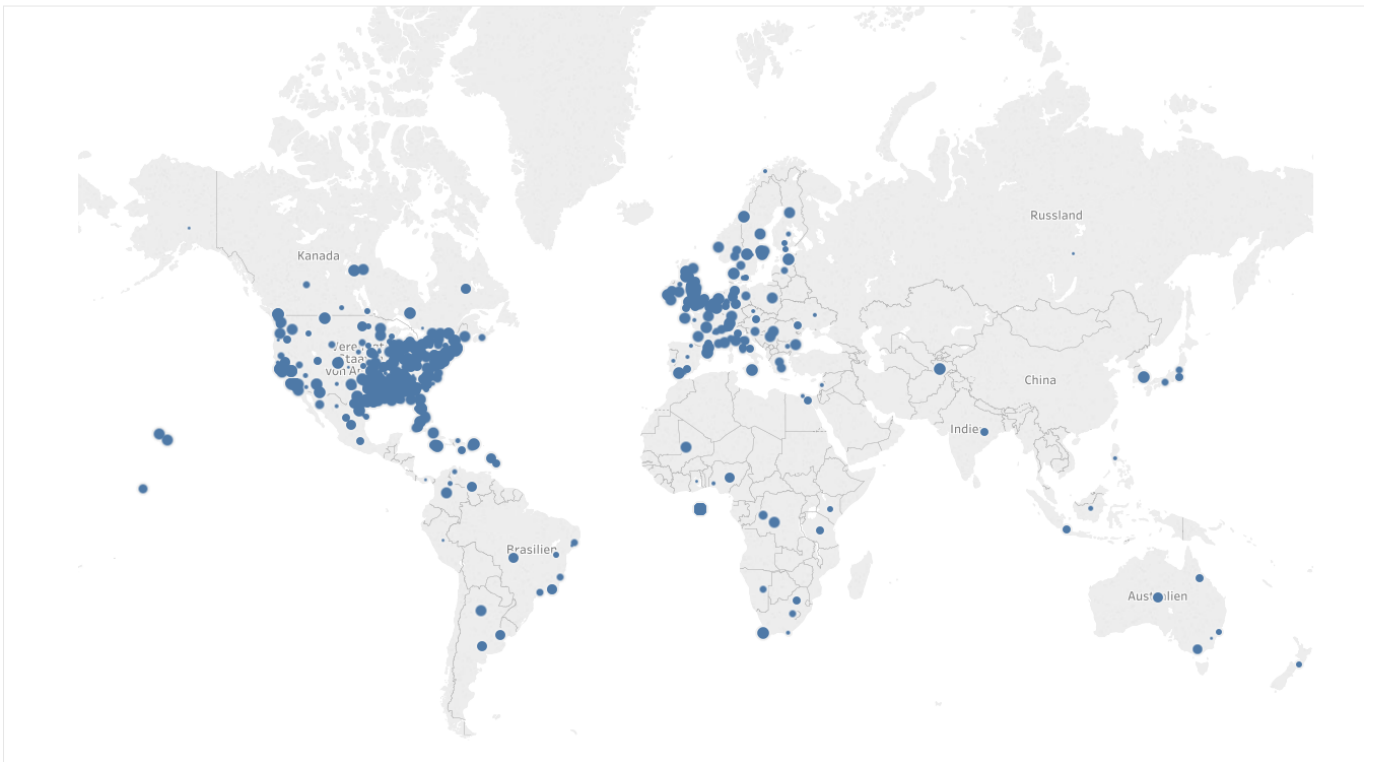
Weitere Fragestellungen

1. Woher kommen die meisten Künstler?
2. Sind die Lieder im Laufe der Zeit kürzer oder länger geworden?
3. Sind schnelle Songs beliebter als langsame Songs?

Zu 1.)

Wir sehen hier, dass viele Künstler aus den USA und Nord/West-Europa liegen. Auffällig ist, dass Asien (Russland, China, Indien) trotz hoher Bevölkerungszahl in diesen Daten fast gar nicht vertreten ist. Wurden hier eventuell nur englische Lieder in der Datenbank eingetragen?

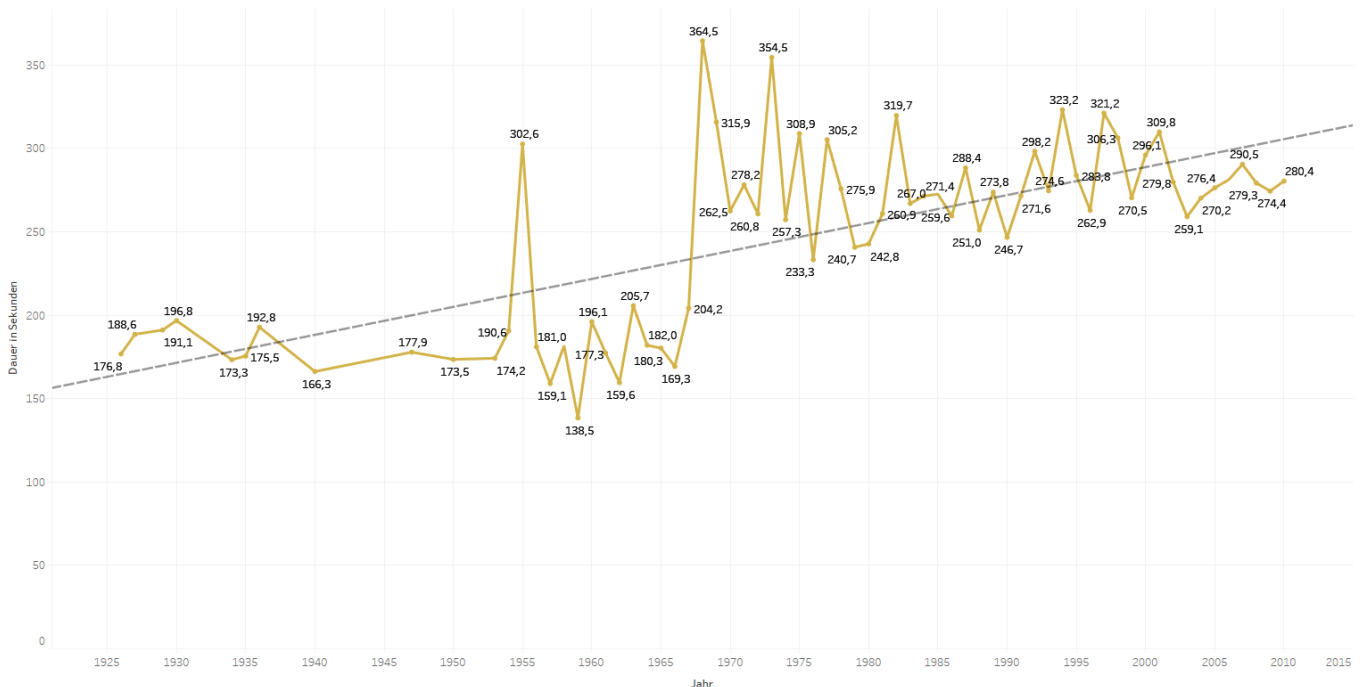
Künstler Herkunft



Zu 2.)

Tatsächlich scheint es, dass im Laufe der Zeit die (durchschnittliche) Länge der Lieder zugenommen hat, sich aber während der letzten Jahrzehnte etwas eingependelt hat bei ca. 280 Sekunden (4 Minuten, 40 Sekunden)

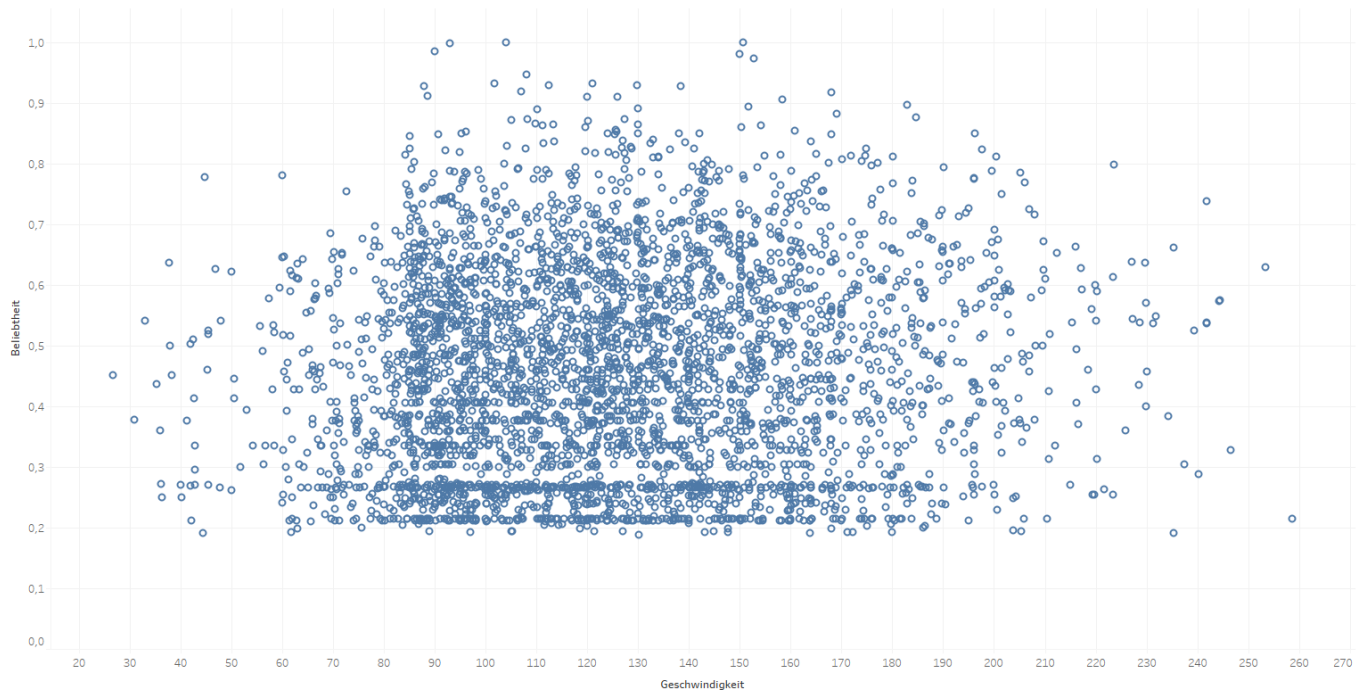
Liedlänge im Laufe der Jahre



Zu 3.)

Leider lässt sich die Frage nur schwer beantworten. Es scheint logisch, dass weder zu langsame, noch zu schnelle Songs zu den beliebtesten zählen. Eine Tendenz lässt sich eher nicht erkennen. Auf dieser Abbildung entspricht jeder Punkt einem Songtitel:

Schnell vs. Ruhig



Aufgabe 3

a)

Für das Binning haben wir 10 bins gewählt, diese mit PySpark erstellt und exemplarisch die Tabelle zeigen lassen. Wir haben festgestellt, dass der Bucketizer binning betreibt, indem dieser eine Spalte hinzufügt, in der die Zuordnung zu einem bin steht.

```
+-----+-----+-----+-----+
|          title|          track_id|timbre_0|buckets|
+-----+-----+-----+-----+
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  21.613|    2.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  21.864|    3.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  20.972|    2.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  22.255|    3.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|   21.82|    3.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  21.105|    2.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  21.605|    2.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  29.292|    4.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  36.902|    5.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  35.457|    4.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  30.749|    4.0|
|Doppelgoonger [Ql...|TRABEFN128F92D925B|  31.128|    4.0|
```

Gespeichert haben wir die Tabelle dann als `mi6xc_bucketeddata` und zur Sicherheit den Speichervorgang überprüft.

```
%pyspark
df = spark.sql("select * from mi6xc_bucketeddata limit 5")
df.show()
```

```
+-----+-----+-----+-----+
|          title|          track_id|timbre_0|buckets|
+-----+-----+-----+-----+
|Doppelgoenger [Ql...|TRABEFN128F92D925B| 21.613| 2.0|
|Doppelgoenger [Ql...|TRABEFN128F92D925B| 21.864| 3.0|
|Doppelgoenger [Ql...|TRABEFN128F92D925B| 20.972| 2.0|
|Doppelgoenger [Ql...|TRABEFN128F92D925B| 22.255| 3.0|
|Doppelgoenger [Ql...|TRABEFN128F92D925B| 21.82| 3.0|
+-----+-----+-----+-----+
```

Anschließend haben wir die Pivotierung durchgeführt.

```
%pyspark
#aufgabe 3b
from pyspark.sql.functions import count
df = spark.sql("select * from mi6xc_bucketeddata")
piv_df = df.groupBy('track_id').pivot('buckets').avg('timbre_0')
```

```
#Beispiel zum Umbenennen der Spalten aufgrund von https://issues.apache.org/jira/browse/SPARK-12965
#df = spark.sql('show tables')
#print(df.columns)
#newNames = ['track_id', 'bin1', 'bin2']
#newdf = df.toDF(*newNames)
#print(newdf.columns)
```

```
# Umbenennen der Spalten
new_names = ['track_id', 'bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'bin_5', 'bin_6', 'bin_7', 'bin_8']
new_df = piv_df.toDF(*new_names)
new_df.show()
```

```
new_df.write.mode("overwrite").saveAsTable("mi6xc_profilevectors")
```

```
+-----+-----+-----+-----+-----+-----+
|          track_id|          bin_0|          bin_1|          bin_2|          bin_3|          bin_4|
|          bin_5|          bin_6|          bin_7|          bin_8|
+-----+-----+-----+-----+-----+-----+
|TRAADQX128F422B4CF|2.6020000000000003|11.431999999999999| 20.41790909090909| 24.7179880952381|33.824558333333314|
| 39.56649816849816|46.288232104121434|          null| null|
|TRBCJKE12903CE3A8B|          0.0| 9.273|          null|          null| 33.255625|
|41.380091787439596| 45.86282278481014| 50.587| null|
|TRASOQC12903CFFED7|          0.0|          null|          null|          null|          null|
| 40.91746153846154| 48.96741322314051|53.140327413984394| null|
|TRAVGKR128E0789A99|          0.0| 11.116| 19.92592307692308|25.556989795918373| 32.37147328244274|
|38.997725000000024| 44.637999999999999|          null| null|
|TRANTMY128F92D3536|          7.211|          null|          null| 24.779|34.188250000000004|
| 41.31847058823531|47.420485094850946| 52.26062903225807| null|
|TRATPUS128F4271260|          0.0|          null|          null| 24.465| 34.28797222222223|
| 40.79228421052629| 46.0264318766067| 50.977| null|
```