UNIVERSITY OF PISA

MASTER'S DEGREE IN
DATA SCIENCE & BUSINESS INFORMATICS

DISTRIBUTED DATA SCIENCE AND MINING

SPOTIFY ANALYSIS

ALESSANDRO BONINI[604482]
CARLO ALBERTO CARRUCCIU[533967]
PASQUALE GORRASI [587417]
GIAN MARIA PANDOLFI[607268]

October 27, 2021

Contents

1 Introduction

1.1 Objectives

The project is developed on a two *"csv"* dataset available on the *Kaggle platform*. The data have been obtained by **Spotify**. Te main one, *"track.csv"* the most important and largest, contains music tracks informations from a period of 100 years. The other one instead, *"artist.csv*",contains a row for each artist. Basing on the suggestion of the dataset's author, we identified three main analysis to apply on the data :

- **Clustering**: on the songs, to identify a limited number of genres
- **Classification/Regression**: to understand which are the most important features in estimating the popularity of a song
- **Trend Analysis**: to see how musical creation changed above the years

1.2 Background

The project requirement is to use **pyspark** for all the analysis. The idea is to behave as if data are too huge to be analysed within an environment like pandas. However it is still possible to use other libraries only on big aggregations, so after data granularity and size has been reduced considerably. To develop the project, all members installed pyspark and Hadoop on their personal computer, running the notebook on local nodes, or in some occasions, in Google Colab.

2 Data Understanding

In this section we'll present how we've studied, cleaned, integrated and explored our datasets.

2.1 Data Cleaning

Both datasets have been cleaned of null values by deleting the records that presented any of them, given their small number of them compared to the total number of records. In second instance, we observed that the tracks dataset was presenting unexpected data for some records as shown in 2.1.

explicit
687600
Hoffmann
42507
Melot)"
...

Table 2.1: unexpected data in *explicit* column

2.1.1 Tracks

The track.csv dataset is made up of 586672 records defined by 20 attributes. Originally the attributes are all of type string by default, so the first operation done on the dataset was to apply a cast to them as shown in the table. 2.2.

Feature Name	**Data Type**	**Description**
id	*string*	track id, primary key of the table
name	*string*	track's name
popularity	*integer*	popularity of the track measured on a 0-100 scale by an algorithm
duration_ms	*integer*	track duration in milliseconds
explicit	*integer (boolean)*	presence of explicit language on the track
artists	*string*	names of the artists that composed the track
id_artists	*vector of strings*	id of the artists that composed the track
release_date	*date*	publication date
danceability	*float*	measured in range 0-1. Describes how well suited a song is to dance based on a combination of musical elements including tempo, rhythm stability, beat tenor and regularity
energy	*float*	represents a perceptual measure of intensity and activity
key	*integer*	Denotes the major or minor scale in which a piece of music operates, and therefore, the notes that are part of it
loudness	*float*	audio volume measured in LUFS (Loudness Units relative to Full Scale)
mode	*Integer*	tonality(major or minor) of a track
speechiness	*float*	measure in range 0-1, detects the presence of spoken words in a track. Values greater than 0.66 describe tracks made up entirely of spoken words.
intstrumentalness	*float*	This value represents the amount of voices in the song. The closer it is to 1.0, the closer the song is instrumental.
acousticness	*float*	A 0-1 confidence measure that measures how acoustic the trace is
liveness	*float*	This value describes the probability that the song was recorded with a live audience. According to the official documentation: "a value greater than 0.8 gives a strong probability that the track is live"
valence	*float*	A measure from 0.0 to 1.0 that describes the musical positivity conveyed by a track. Songs with high valence sound more positive (e.g. cheerful, cheerful, euphoric), while songs with low valence sound more negative (e.g. sad, depressed, angry)
tempo	*float*	measured in BPM
time_signature	*integer*	The time signature (meter) is a notation convention for specifying how many beats there are in each bar (or measure).

Table 2.2: Columns of the Tracks table

The cast produced new null values, this behaviour was expected by the fact that some wrong data was present in the dataset. We've removed again null values and we observed that unexpected data was anymore present as in table 2.1.

explicit
1
0

Table 2.3: correct data in *explicit* column

2.1.2 Artists

The same procedure was applied to artist.csv which presents 1104349 records and 5 attributes:

Feature Name	Data Type	Description
id	*string*	artist id, primary key of the table
followers	*integer*	number of artist's follower on the platform
genres	*Vector of strings*	musical genres of the artis
name	*string*	artist's name
popularity	*integer*	popularity of the artist measured on a 0-100 scale by an algorithm

Table 2.4: Columns of the Artists's table

2.2 Data Integration

After data cleaning, our goal was to join the two dataset. Firstly, we exploded the artist_id column obtaining a row for each artist of the song. Then, we collected all the genres for each song. Hence, during the join operation, for each artist of a song, genres of different artist have been collected as a list of vector and then flattened in a single vector, obtaining this attribute for each song. Moreover, we considered useful to create new attributes based on the **popularity** of the artists and on the **number of followers** they have; with this purpose *"sum"* and "*average"* aggregation functions have been used on these two attributes during the join. The code in the snippet below has been used for this operation. As last step we performed a select distinct to remove possible duplicates.

```
windowSpec = Window.partitionBy("id_track")

tracks_df_wk2 = tracks_df_wk1.join(artist_df, tracks_df_wk1.id_artist==artist_df.id,"left") \
            .filter(col("popularity").isNotNull()) \
            .filter(col("followers").isNotNull()) \
            .withColumn("sum_artist_followers",sum(col("followers")).over(windowSpec)) \
            .withColumn("sum_artist_popularity",sum(col("popularity")).over(windowSpec)) \
            .withColumn("avg_artist_followers",F.avg(col("followers")).over(windowSpec)) \
            .withColumn("avg_artist_popularity",F.avg(col("popularity")).over(windowSpec)) \
            .withColumn("collect_list_genres", collect_list("genres").over(windowSpec)) \
            .withColumn("collect_list_genres", flatten(col("collect_list_genres"))) \
            .withColumn("collect_list_genres", array_distinct("collect_list_genres")) \
            .withColumn("genres", array_remove("collect_list_genres", "")) \
            .drop("collect_list_genres")
```

2.3 Data Exploration

2.3.1 Correlation

During this phase we focused on our goals. In general it was important for all the tasks to look at the correlation between the dataset, so in figure 2.1 is shows an heat-map of the correlations between attributes. The main scope is to find some attribute that should be discarded during the clustering [1], but also to have already an idea of the impact of the features over the popularity of the songs.

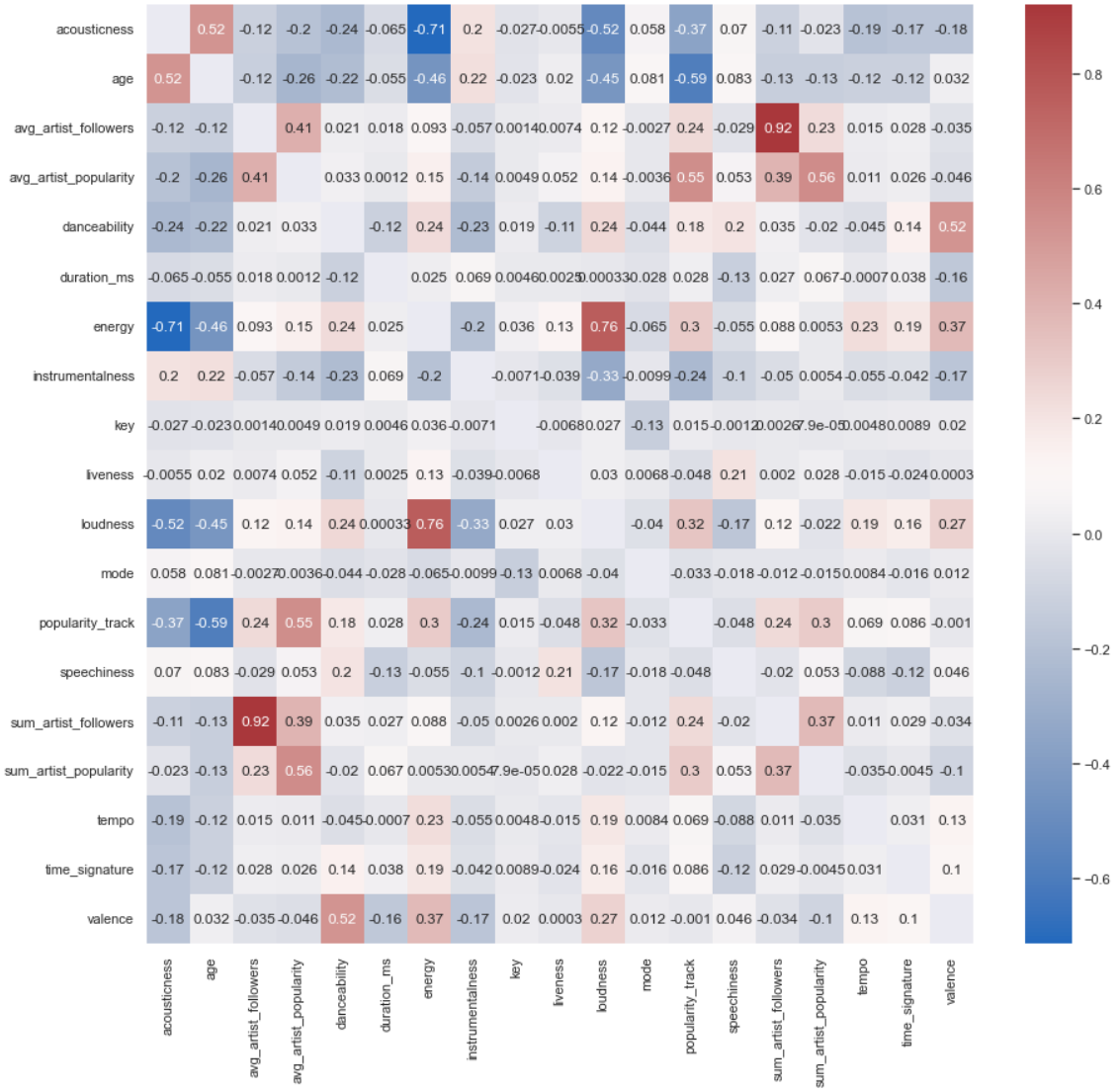


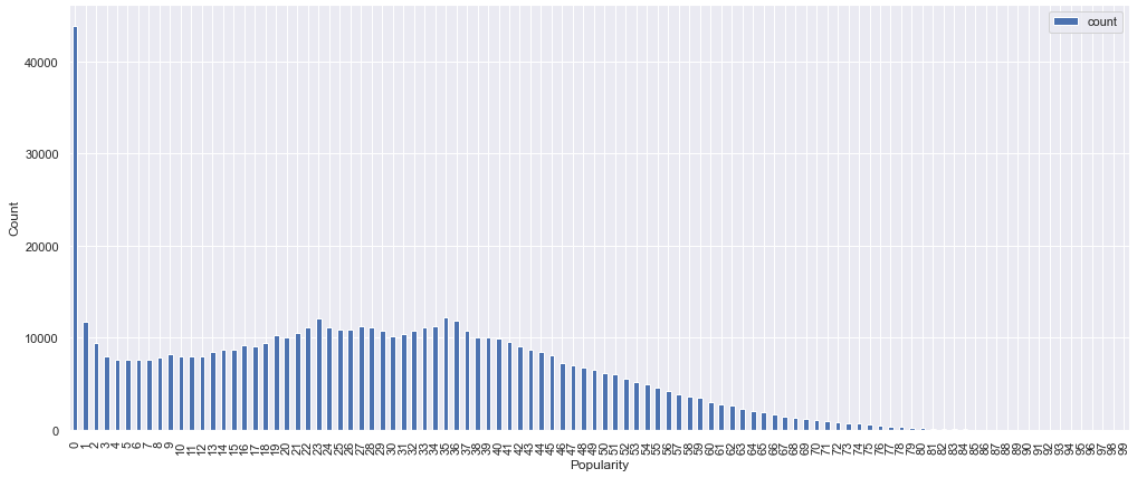
Figure 2.1: Correlation

Different useful informations, emerged from this analysis. In the optic of making the clustering, we can for example pay attention on the attribute *"energy"*, highly correlated both with "loudness" and *"acousticness"*, so a candidate to be discarded. As expected, information extracted by the artist's dataset (average and total popularity/followers) can be really decisive in predicting the popularity of the song, in particular the average popularity of the artists. The attribute age is correlated with a lot of attributes, meaning that there are trends above years for the song's characteristics.

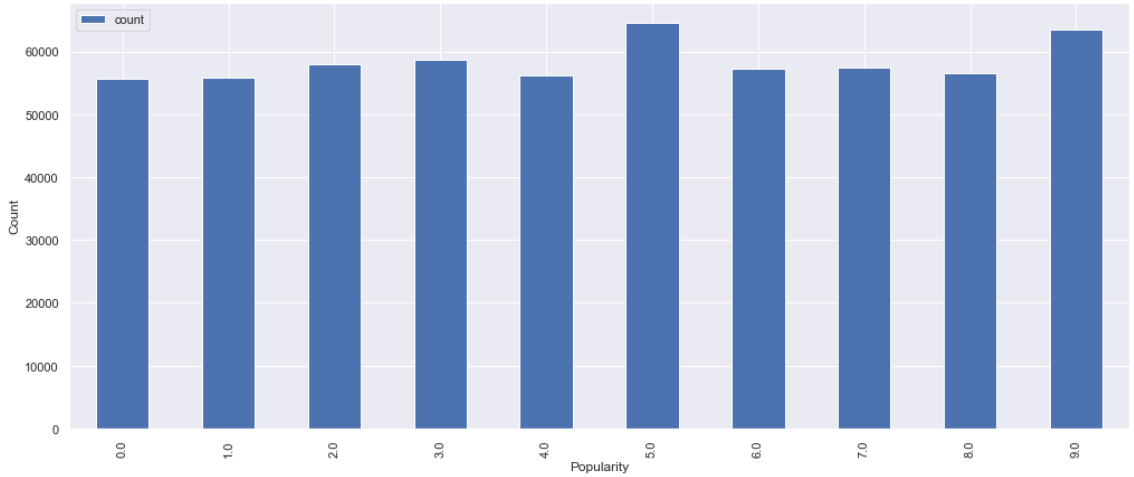
[1] Attributes with an high correlation tend to polarize the most of clustering algorithms

2.3.2 Distributions of data

We focused on one distribution in particular, the one interesting for the classification-regression task. The distribution is not uniform and is totally unbalanced, tending to the 0 value of popularity. However what we observed is that we can easily balance it in class, using the ***"quantile discretizer"*** provided by pyspark. In figure 2.2 are shown the distribution of records over popularity, and the distribution of records over popularity classes created by discretization.



(a) Popularity Distribution



(b) Popularity Class Distribution

Figure 2.2: Popularity Distributions

3 Clustering

Given the high number of genres (over 4000), the goal for this task was to reduce the present genres to a small number of macro-genres. Clustering is the ideal action to solve this kind of problems. The goal of clustering is to find an intrinsic structure in the data, so that each trace of a macro-genus is closer to another trace of the same macro-genus (lower *intracluster* distance) and further away from another. member in a different macro-genus (greater *intercluster* distance).

Pyspark provides, on its machine learning library, implementations of the Kmeans and of the bisecting kmeans. Moreover to apply the hierarchical clustering we used scikitlearn, after having grouped song by genre. So these three algorithm have been used:

- **K-means**
- **Bisecting K-means**
- **Agglomerative Clustering** (scikit learn)

3.1 K-means and Bisecting K-means

The first operation is to add the 'features' attribute to the records, constructed as a vector composed of the numeric values of all the other attributes. This attribute allows to identify the records in a Cartesian space on which the k-means will be applied. Among the attributes chosen to compose this vector, categorical ones and redundant numeric attributes have been eliminated, for a total of 16 attributes used.

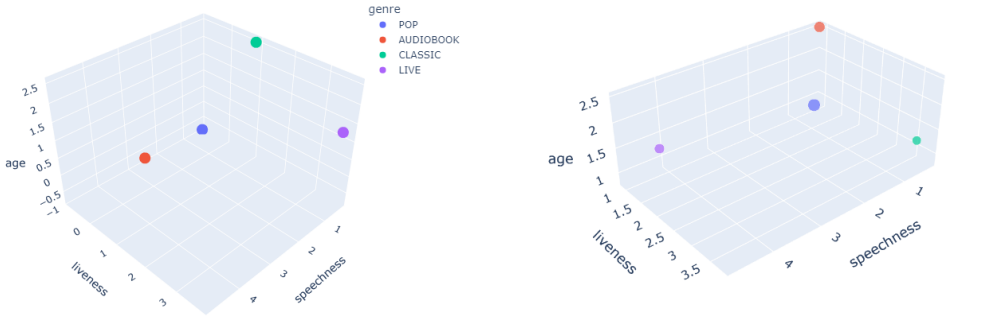
Before starting the algorithm it is necessary to reduce the dimensionality of the features attribute, keeping only the values that capture the greatest variance, in order to obtain qualitatively better results. To do this, the ***pyspark Variance Treshold*** was used; so attributes with variance less than 1 were eliminated from *'features'*. This left us with 3 attributes for each record, specifically *"speechiness"*, *"liveness"*, *"age"*.

Finally, to choose the optimal number of clusters, the silhouette score for different clusters (from 2 to 10) was evaluated, the evaluation suggests a number of clusters between 4 and 6, after evaluating the results we opted for 4 clusters (silhouette = 0.61).

Clusters centers, shown in figure 3.1 clearly link the musical tracks to the following genres:

- **POP**: with centroids = [0.40061972 - 0.86353734 - 0.79206732] indicating a low speechiness value (little spoken), a low probability of live recording (therefore studio sounds, e.g. electronics) and a very recent track release date.
- **AUDIO BOOKS**: with centroids = [4.84705963 2.05296189 2.03091133] with a very high speechiness value which indicates the high density of speech within the track and average values of age and liveness, not decisive.
- **CLASSIC**: with centroids [0.33774966 0.91381392 2.54167034] which indicate a low value of speechiness and liveness and a high value of the age attribute, demonstrating the time passed since the release date.
- **LIVE**: with centroids [0.51305821 3.71835118 1.3537786]. This last cluster gathers several genres united by a high liveness attribute value to indicate the live recording of the track.

The second clustering used is bisecting k-means, again using the spark library. Unlike the previous one, it is a top-down clustering, which starts from a single cluster and performs divisions up to the desired number of clusters. Also in this case the evaluation was carried out by comparing the silhouette values, and also in this case the optimal result is 4 clusters (silhouette =



(a) Kmeans Clusters Centers

(b) Bisecting Kmeans Clusters Centers

Figure 3.1: Clusters Centers

0.61). The result is the following, very similar to k-means.

- **POP:** with centroids = [0.38649703 0.89038973 0.79395606]
- **CLASSIC**: with centroids = [0.32490673 0.92299514 2.52324765]
- **AUDIOBOOK**: with centroids = [0.6498445 3.8400005 1.41456087]
- **LIVE**: with centroids = [4.74552642 2.02605544 2.02572495]

3.1.1 Agglomerative Clustering

Cause agglomerative clustering was not available in pyspark, we need to find an alternative approach. An implementation of this algorithm is provided by *scikit learn*, but we need to reduce the size of our dataset to be able to use this library. The idea has been to calculate the cluster on a aggregation of the dataset by genres. In order to do this two lines of code were necessary, correspondent to the following two steps:

1. Explode the dataset by the column genres, creating the column genre.
2. Group By the column genre using the aggregation function average on the numerical columns.

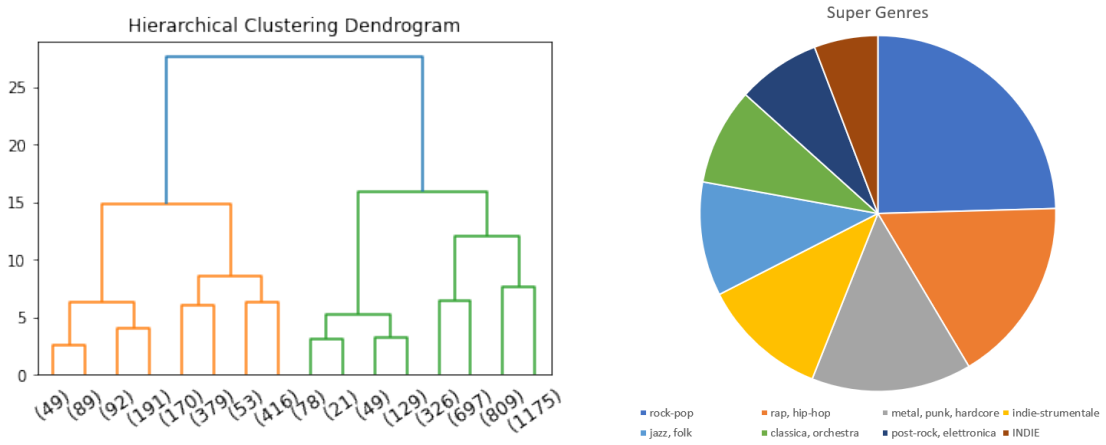
```
 #EXPLODE GENRES
df = df.withColumn('genre',explode(df.genres).alias("genre"))

#GROUP BY GENRE
dfg = df.groupBy('genre').avg('duration_ms', 'danceability', 'energy', 'key' ...

...

#GROUP BY CLUSTER and COLLECT GENRES
df = df.select('cluster','genre').groupBy('cluster').agg(collect_set('genre').alias("genres"))

#CONCAT GENRES TO A STRING
df = df.withColumn('genres', concat_ws(" ",col('genres')))
```

Listing 1: Explode and group by Genre

During transformation we passed from more than 500.000 rows of the cleaned dataset, to over

2,5 millions of row in the exploded dataset, to only 4.000 rows in the grouped dataset, one for each genre.

After that, was possible to proceed with the hierarchical clustering. Best result have been obtained using euclidean distance as metric and the ward linkage [2]. From the dendrogram in figure 3.2a we opted to take 8 different genres.



(a) Dendrogram

(b) Distribution of genres over super-genres

Figure 3.2: Hierarchical Clustering

At this point other transformation were done using pyspark function: the dataset has been grouped by the cluster assigned, hence it was possible to collect all sub-genres in a vector column; then we applied a transformation to these columns concatenating all the genre in an unique string.

So for each cluster most frequent words have been retrieved and we used these words to infer the macro genres represented in the pie chart at figure 3.2b.

[2] Other metrics and linkage methods failed in differentiate genres in balanced clusters

4 Regression

The popularity was the target attribute of our analysis, having integer values from 0 to 100; however we created ten popularity classes to balance the attribute, how shown in figure 2.2b. After different attempts, we discovered that the regression performs better on the initial popularity, so the quantile discretizer is used during the evaluation phase.

Our main goal is to use the two algorithms selected from pyspark (linear regressor and RandomForest Regressor) to evaluate the importance of each features to determinate the target.

4.1 The Attributes

Before performing the regression we isolated the strictly technical attributes (*'acousticness',' danceability ',' duration_ms', 'energy', 'instrumentalness',' key ',' liveness', 'loudness',' mode ',' speechiness ', 'time', 'time_signature', 'valence', 'popularity_track'*) to evaluate their influence on the target attribute 'popularity'. We were curious to see how these attributes impacted popularity without distractions from other attributes. So we can see which are the valuable techniques to make a popular song.

Moreover we trained the regression using in addition to the previous features mentioned, the calculated attributes too, such *'age'* and the ones obtained by the artist table (total and average popularity/follower). We know that these attributes could be particularly relevant in predicting.

4.2 Regression

At the beginning results were not really satisfying, returning a low r2 score[3] for both the regressors. So, we processed the features used: the *MinMaxScaler* has been applied over all the features. The pre-processing step applied shown an evident increase of the regression performance (see table 4.1). Using only the technical attributes, the coefficient of determination is now 0.2 for the linear regression, and 0.26 for the random forest, still low but acceptable. While using all the features, r2 score reach a similar value of 0.53 for the linear regressor and 0.55 for the random forest.

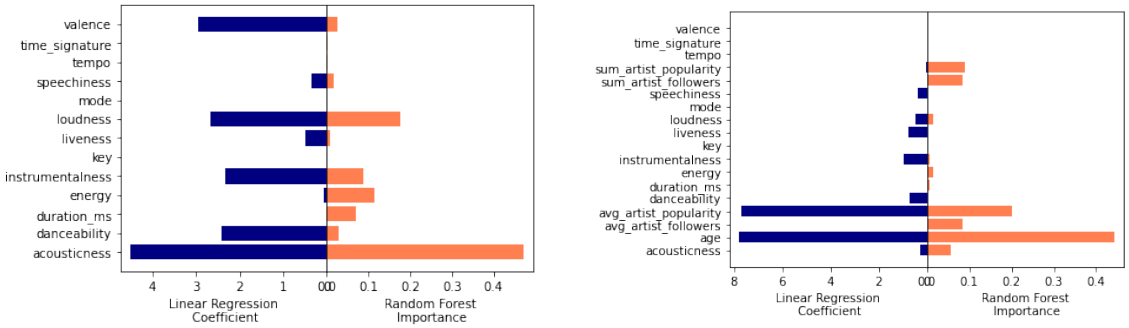
	Technical Features	**All Numerical Features**
Linear Regression	0.20	0.53
Random Forest	0.26	0.55

Table 4.1: Coeffecient of determination (R2 score)

[3] All evaluations have been obtained splitting the data 70% in the training set and 30% in the test set

4.3 Feature Importance

Using these two regressors have been possible to estimate the importance of each attribute. In fact we can use both the coefficients from the linear regression, rather than the feature importance provided by the random forest model[4]. We can inspect all the metrics from the plots in figure 4.1.
In the second plot(4.1b it is evident how the "non technical" features polarized the decision. In particular, the popularity seems to be linearly dependent by "age" and "avg_artist_popularity" ; while random forest use also the other "artist features". Moreover the attribute *"acousticness"* should be mentioned for its importance.
In the second plot(4.1a) only technical attributes have been considered. Here we note acousticness is effectively the most important attribute. While in linear regression other four attributes contributed to the prediction; the random forest gave proportionally little importance to other features, probably only the *"loudness"* should be mentioned.



(a) Only technical attributes

(b) All numerical attributes

Figure 4.1: Feature Importance

[4]Random Forest uses bootstrap sample both on rows, and normal sample on columns, so is really appropriate for this purpose

5 Classification

5.1 Models

In this phase of the project we used different classification techniques to determine the popularity class of a song using the attributes available to us. We used different models by dividing the target attribute, that is *"popularity_track"*, in three different ways:

- 3 bins equally distributed;
- 3 bins not equally distributed;
- 10 bins equally distributed (see 2.2b).

We can divide the used models into two categories, the first concerns multi-class classifiers, such as **MLP**, **naive bayes** and **random forest**, while the second concerns binary classifiers adapted through the use of the **one vs others technique**; for this purpose we have chosen the logistic regression and linear support vector machine. The attributes used for this phase were all numerical attributes already used in regression 4.1.

5.2 Pre-Processing

In the pre-processing phase we have discretized the target attribute, *"popularity_track"* and we have standardized the values contained in the attributes we used for this phase using the *StandardScaler* function. In using the naive bayes, linear support vector machine and logistic regression models for the selection of the most significant attributes we used the *VarianceThresholdSelectior* function. For the two remaining models no feature selection was made as it is not necessary. The dataset was finally divided into 70% for training the model and 30% for testing.

5.3 Results

	Naive Bayes	**Random Forest**	**MLP**	**LSVM**	**Logistic Regression**
Accuracy	0.21	0.35	0.29	0.23	0.24
F1 score	0..20	0.33	0.25	0.15	0.19
Recall	0.58	0.79	0.66	0.81	0.73
Precision	0.41	0.74	0.68	0.33	0.40

Table 5.1: Performances of the classification models over 10 bins equally distributed

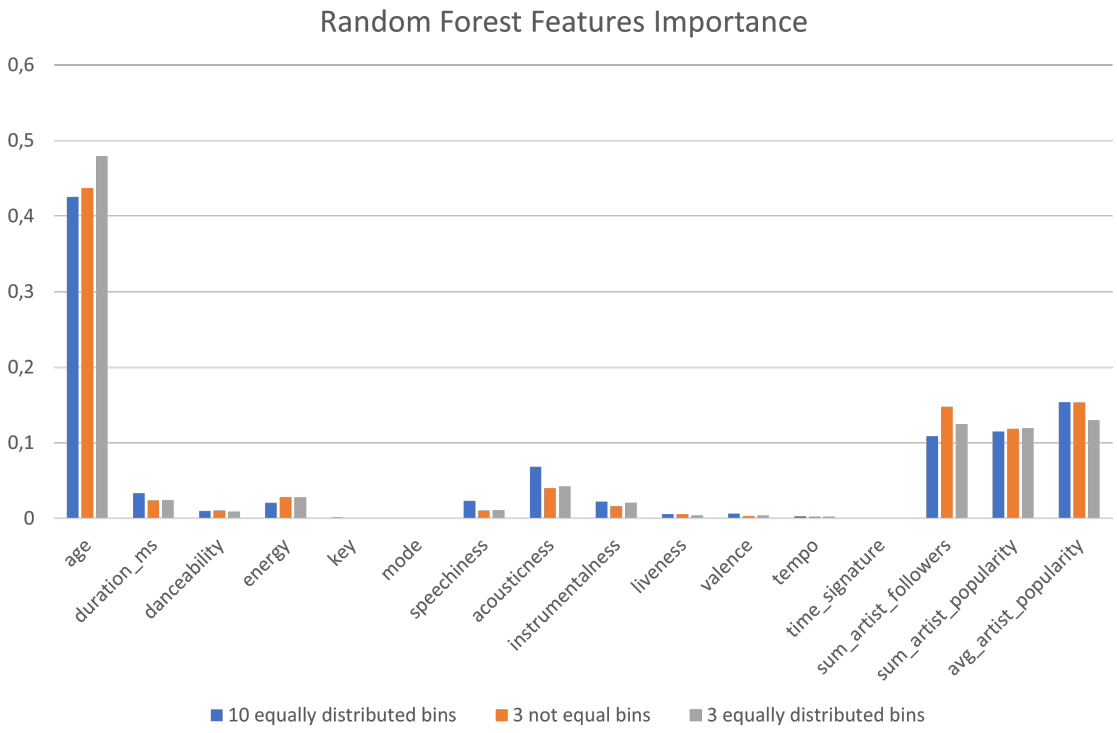
	Naive Bayes	**Random Forest**	**MLP**	**LSVM**	**Logistic Regression**
Accuracy	0.67	0.79	0.77	0.66	0.70
F1 Score	0.67	0.78	0.76	0.62	0.69
Recall	0.81	0.87	0.84	0.93	0.81
Precision	0.73	0.82	0.81	0.67	0.75

Table 5.2: Performances of the classification models over 3 unbalanced bins

We can see in tables 5.3, 5.2, 5.1, without big surprises, that the best performances are given by the classifiers that use a division of the target attribute in three parts not equally distributed. Taking instead the single classifiers we can see that in all three cases the Random Forest is the best performing model followed at short distance by the Multilayer Perceptron, which was set with two hidden layers of 10 and 3 nodes.

	Naive Bayes	Random Forest	MLP	LSVM	Logistic Regression
Accuracy	0.41	0.68	0.65	0.35	0.40
F1 Score	0.33	0.68	0.65	0.23	0.30
Recall	0.90	0.70	0.70	0.97	0.90
Precision	0.44	0.95	0.72	0.37	0.45

Table 5.3: Performances of the classification models over 10 bins equally distributed



Finally, we can see that the most significant attributes used by Random Forest are the features created by us in the preprocessing phase as "age", "avg_artist_popularity", "sum_artist_popularity", "sum_artist_followers", instead among the attributes provided initially by the dataset stands out without a doubt "acousticness", followed by "duration_ms" and "energy".

6 Trend Analysis

This analysis is like the bonus track of our LP: by using some transformation with pyspark, we created a new dataset that results like a set of time series, one for each technical that attribute appears in our initial dataset.

```
from pyspark.sql.window import Window

df = df.withColumn('year',2021 - df.age.cast(IntegerType()))

windowSpec = Window.partitionBy("year")
df = df.withColumn("popularity_norma",sum(col("popularity_track")).over(windowSpec))

dfn = reduce(lambda memo_df, col_name: memo_df.withColumn(col_name+'_n',\
    (col(col_name)*col('popularity_track')/col('popularity_norma'))  ),technical_columns,df)

dfg = dfn.groupBy('year').agg(\
    F.avg('acousticness').alias('avg_acousticness'),\
    F.sum('acousticness_n').alias('wavg_acousticness'),\
    ...
```

Listing 2: Code for calculating the trends

Starting from the attribute *"age"*, the year has been retrieved; it has been used as the grouping attribute for our transformation. However, we decided to use a custom aggregation system. we didn't want an average per year, but we wanted a weighted average per year using popularity as weight of the song. To do this the column year has been used to create partitions and obtaining the sum of popularity over the partition by year. Then, the column popularity has been divided for this sum, obtaining a sort of coefficient. After that, each column has been multiplied for this coefficient and in the end we are ready to aggregate by sum all of these normalized columns grouping the dataset by year. A sample of the obtained dataset is reported in table 6.1.

year	*min_acousticness*	max_acousticness	avg_acousticness	wavg_acousticness	min_danceability	max_danceability	avg_danceability	wavg_danceability	min_duration_ms	max_duration_ms	...
1972	0.000007	0.996	0.546940	0.022653	0.0000	0.921	0.513294	0.021259	31453	4142067	...
1973	0.000004	0.996	0.514682	0.020233	0.0000	0.942	0.518486	0.020383	14708	2052573	...
1974	0.000003	0.996	0.522935	0.022950	0.0000	0.946	0.517479	0.022711	30067	3153173	...
1975	0.000005	0.996	0.497883	0.021566	0.0641	0.947	0.519428	0.022499	31280	3091707	...
...	...	...	...	...	...	...	...	...	...	...	...

Table 6.1: Time Series Dataset

Once this dataset was created, each technical feature has been plotted in figure 6.1 as a time series for the last 50 years. First thing we can observe is that average and weighted average trends coincide in all of cases, except for the attribute *"instrumentalness"* ; this attribute in the last 20 years is quite lower in popular songs respect to the average. In fact, the average of this attribute is increased a lot since the beginning of the new millennium; nevertheless songs to be popular is better to have an important vocal component. All other attributes don't show any difference between the average and the weighted trend. The trends show more or less what was expected by the correlation analysis performed during the data understanding(2.3.1); positive trends coincide with a positive correlation and vice versa negative trends characterize inversely correlated features.

Speaking about the last 10 years we can say that there was a drastic drop of the *"liveness"* and *"mode"*, but most of all, the duration of the song reduced a lot. Ten years ago the average duration of a song was 3 minutes; today it is only 2 minutes. In opposite of these drops, the *"danceability"* in the last 10 years really increased, and it has still not reached is peak.

Looking at longer trends, is curious as *"speechiness"*, after having is lower peak between 2000 and 2010, now is growing up, perhaps due to the advent of audio books. Another interesting feature is the loudness, characterised by a strange S-shaped positive trend. The trend was flat till 90', then the volume of songs exploded till 2010, to became then another time a flat trend, decreasing a bit in the last few years. Other attributes not mentioned, have a linear positive or negative trend, according with their correlation with *"age"*.

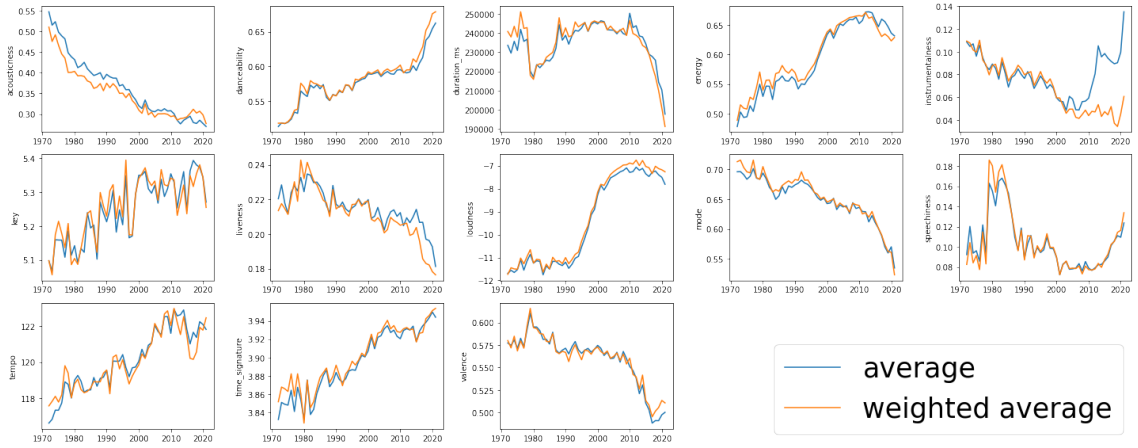


Figure 6.1: Time Series