Rapport projet de visualisation

Cours d'Information Visualisation SI5

Rédigé par :

Khaoula BOUHLAL
Souhail LEBBAR
Kévin CONSTANTIN

Année universitaire 2022-2023

Table des matières

Présentation générale du projet

Objectif

L'objectif du projet est d'amener au mieux la cognition afin de traiter les données pour la visualisation. La data est une représentation d'information, à elle seule, elle ne veut pas dire grand-chose. D'où le besoin d'effectuer un traitement qui nous permet d'en faire une information.

Nous utiliserons la data qui elle deviendra connaissance. Cette visualisation nous permettra de déceler et de comprendre des informations, les données brutes étant difficilement interprétable et exploitable

Dans ce projet nous nous sommes intéressés aux musiques contenant des paroles dites « Explicit » (langage grossier). Chaque étudiant analysera ces informations en se concentrant sur un élément en particulier.

Données utilisées

Les données qui ont été utilisés sont celles du dataset wasabi, provenant du github du projet disponible ici. Les données des artistes, albums et musiques ont été utilisés (téléchargé) dans leur intégralité (77k artistes, 208k albums, et 2.1M de musiques) avant quelque traitement que ce soit.

L'extraction et la transformation des données est un point essentiel. C'est cela qui nous permettra de bien communiquer. Notre projet a été orienté ainsi :

Données brut ➡ Importation ➡ Nettoyage des données+ enrichissement ➡ mapping -->(Comment représenter cela, couleur, forme…)

Opérations effectuées sur les données

> **"Torture the data, and it will confess to anything."**
> **— Ronald Coase**

En se basant sur ce principe, on a vu qu'afin d'avoir de meilleures visualisations il a été primordial de faire un prétraitement plus fin et pointu de toute la base de données se trouvant dans le Git WASABI, soit 3 millions de lignes et plus de 100 colonnes au total.

Afin de relever ce défi, le prétraitement avec R seul est trop long et peut entrainer une grosse perte de temps, par exemple, juste en chargeant ou déchargeant les données ça nous prenait plusieurs minutes. D'où par conséquent le choix d'utiliser une nouvelle technologie, **Spark.**

Avant d'enchainer sur l'explication détaillée de notre prétraitement, on vous présentera l'outil utilisé ainsi que les différents avantages majeurs que présente cette technologie par rapport à R studio.

Présentation de Spark :

Les applications Spark se composent d'un processus pilote et d'un ensemble de processus exécuteurs. Le processus du pilote exécute votre fonction main(), se trouve sur un nœud du cluster et est responsable de trois choses :

1. Maintenir les informations sur l'application Spark ;
2. Répondre au programme ou à l'entrée d'un utilisateur ; et analyser, distribuer et programmer
3. Travailler à travers les exécuteurs (défini momentanément). Le processus du pilote est absolument essentiel - c'est le cœur d'une application Spark et conserve toutes les informations pertinentes pendant la durée de vie de l'application.

Les exécuteurs sont responsables de l'exécution effective du travail que le conducteur leur assigne. Cela signifie que chaque exécuteur n'est responsable que de deux choses : exécuter le code qui lui est attribué par le conducteur et rapportant l'état du calcul, sur cet exécuteur, retour au nœud du pilote.

Maintenant, alors que nos exécuteurs, pour la plupart, exécuteront toujours du code Spark. Notre chauffeur peut être « conduit » depuis un certain nombre de langues différentes via les API de langue de Spark.

API de Spark :

Les API de langage de Spark vous permettent d'exécuter du code Spark à partir d'autres langages. Pour la plupart, Spark présente quelques « concepts » de base dans chaque langue et ces concepts sont traduits en code Spark qui s'exécute sur le cluster de machines.

- ***SCALA*** : Spark est principalement écrit en Scala, ce qui en fait le langage "par défaut" de Spark. Ce livre inclura des exemples de code Scala chaque fois que cela sera pertinent.
- ***JAVA*** : Même si Spark est écrit en Scala, les auteurs de Spark ont veillé à ce que vous puissiez écrire du code Spark en Java. Ce livre se concentrera principalement sur Scala mais fournir des exemples Java, le cas échéant.
- ***PYTHON*** : Python prend en charge presque toutes les constructions prises en charge par Scala. Ce livre inclura des exemples de code Python chaque fois que nous inclurons des exemples de code Scala et qu'une API Python existe.
- ***SQL :*** Spark prend en charge la norme ANSI SQL 2003. Cela permet aux analystes et aux non-programmeurs d'exploiter facilement les pouvoirs de Big Data de Spark. Ce livre comprendra SQL exemples de code le cas échéant.
- ***R :*** Spark possède deux bibliothèques R couramment utilisées, l'une faisant partie du noyau Spark (SparkR) et l'autre en tant que package piloté par la communauté R (sparklyr). Nous couvrirons ces deux différentes intégrations dans la partie VII : Écosystème.

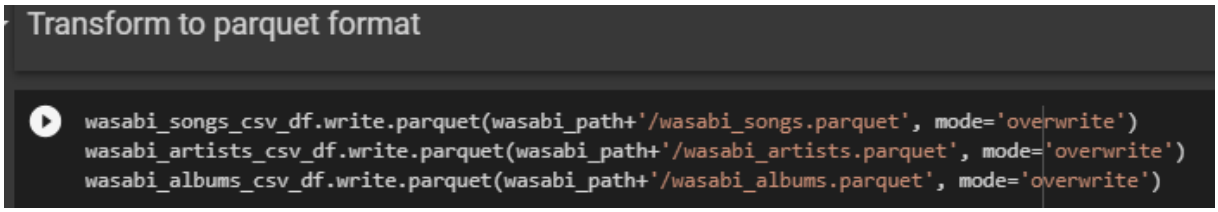
Pour prendre avantage par notre flexibilité de programmation avec Python, on a choisi d'utiliser Python avec l'API PySpark.

Prétraitement de données :

Dans cette partie on se focaliserait plutôt sur les différentes techniques qu'on s'est amusé à utiliser pour faire le prétraitement de données plus que d'expliquer que fait chaque bout de code.

- *Changement de la forme de donnée*

La première est d'utilisé la forme Parquet plutôt que la forme CSV puisque ça nous permet de paralléliser le process d'importation de données.



- *Sélection des bons champs ainsi que l'application des agrégats avec SQL*

Il suffit d'enregistrer la DataFrame dans une vue temporaire SQL et exécuter la requête avec méthode .sql()

```
#join songs table, album table and artist
Query = """
    SELECT Distinct artist_id, name AS artist_name, genres AS artist_genres, type AS artist_type, location_artist_info AS artist_locations,
           album_id, album_title, genre_album as album_genre, country as album_released_country,
           song_id, song_title, explicit_lyrics
    FROM songs_albums
    join artistsV
    USING(artist_id)
"""

[ ] wasabi_explicit = spark.sql(Query)
```

- *Filtrage de table des selon le besoin*

Ex : filtrage sur la table des sons sur la base de la PATTERN ci-dessous

```
# spliting song_id column with pattern to extract the id
PATTERN = '\((.*?)\)'
# Create a new column called song_id based on the second item in splits
songs_df = songs_df.withColumn("song_id",F.regexp_extract(songs_df.id_song,PATTERN,1))
# same thing for id_album
songs_df = songs_df.withColumn("album_id",F.regexp_extract(songs_df.id_album,PATTERN,1))
# format explicit_lyrics
songs_df = songs_df.replace('1.0','True').replace('0.0','False')
# Drop the splits column
songs_df = songs_df.drop('id_song','id_album')
songs_df.limit(10).toPandas()
```

Je vous laisse explorer les restes du processus de prétraitement en ouvrant le notebook se trouvant dans le code fourni avec le rapport.

Projets de visualisations

Pour la visualisation nous avons trois cas possibles pour la donnée explicit lyrics

- Premier cas : la musique est analysée et elle contient des gros mots
- Deuxième cas : la musique est analysée et elle ne contient pas de gros mots
- Troisième cas : la musique n'est pas analysée.

Partie de Khaoula Bouhlal

Cette visualisation consiste en une représentation de la répartition du langage explicite dans la musique à travers le monde.

Pour cette visualisation nous avons eu besoins des données suivantes :

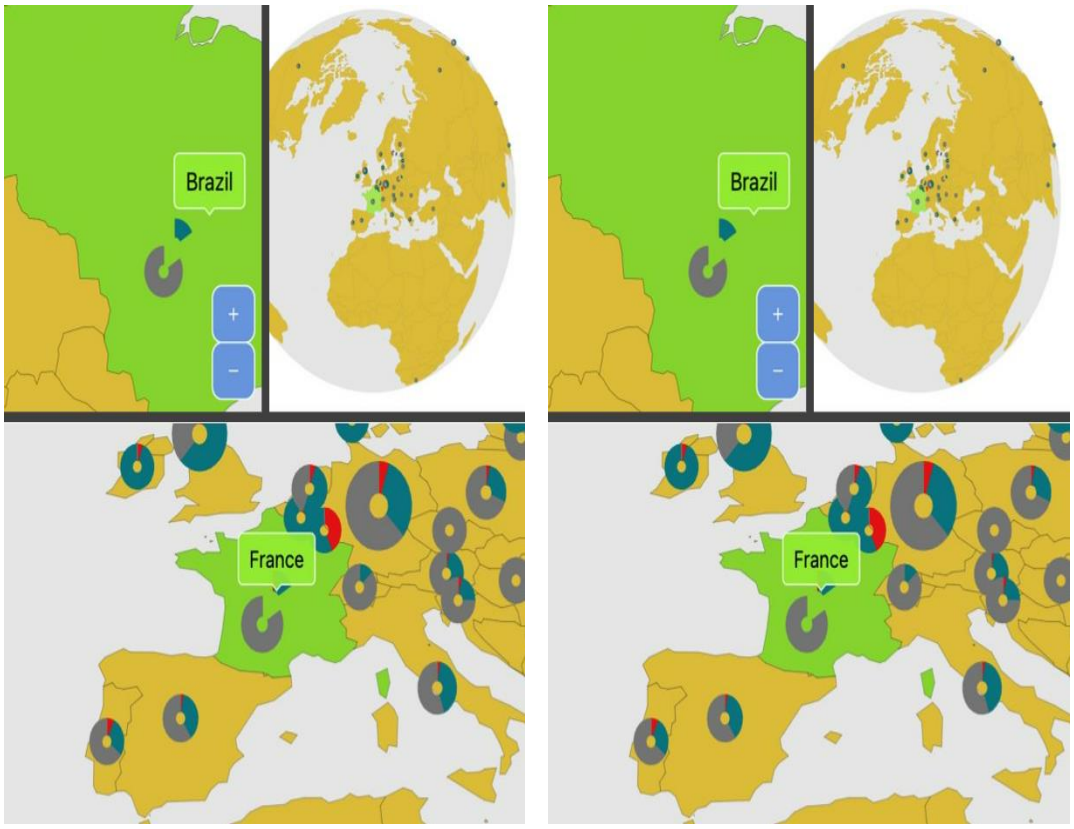
Dans le Dataset :

album_released_country
Explicit_lyrics

Source extérieure :
Country latitude
Country longitude

En effet, le dataset seul ne suffit pas pour placer les graphiques sur les pays. Nous avons donc eu recours à une source extérieure (Kaggle : site de base de données Open Source). Grâce à cela, nous avons pu avoir les coordonnées de chaque pays, à partir de son code à 2 lettres.

Technique de visualisation : Donut maps



Dans un but d'optimisation de la visualisation nous avons optée pour les choix suivants :

- Une vue d'ensemble en 3D pour une meilleure interaction et une navigation plus fluide sans limite de bordures.
- Une possibilité de zoom (avec les boutons graphiques +/- sur le côté, avec la molette, ou en pinçant le pavé tactile).
- Noms des pays visibles lors du click ou passage avec la souris. Changement de couleur pour le pays sélectionné.
- Des détails à la demande : affichage du pourcentage exact de la catégorie lors du passage avec la souris sur le donut. Isolation d'un des quartiers du graphique grâce au click.

Pour créer les graphiques, nous avons fait plusieurs traitements dans le JS. Tout d'abord, nous avons répertorié tous les pays présents dans le dataset, et nous leurs avons assigné leurs coordonnées. Ensuite, pour chaque pays, nous avons calculé le nombre de chansons explicites, non explicites, inconnues et enfin, le total des chansons.

Grâce à ces données, nous pouvons créer un donut chart pour chaque pays répertorié. Sa taille est proportionnelle au nombre total des chansons du pays.

Le côté graphique a été facilité par l'utilisation de la librairie amCharts. Elle était difficile à prendre en main, mais très complète. Notamment, c'était l'une des seules à proposer un globe avec l'intégration de graphiques complets, et non seulement des points sur une carte.

Partie de Souhail Lebbar

Cette visualisation génère un bar chart représentant le caractère des chansons par tranche de 10 artistes.

Pour cette visualisation nous avons eu besoins des données suivantes :
Dans le dataSet:

- Artiste_id
- Artiste_name
- Explicit_lyrics

Technique de visualisation : Bar chart

Dans un but d'optimisation de la visualisation nous avons optée pour les choix suivants :

- L'axe X du graphique montre le nom des artistes. Et l'axe Y montre le nombre de musiques. Chaque barre est découpée en trois : Ses parties représentent le nombre de musique contenant du langage explicite, n'en contenant pas et les musiques pour lesquelles nous n'avons pas de donnée.
- Intégration d'un curseur, pour pouvoir choisir la plage des artistes. Un label au-dessus du curseur affiche la plage sélectionnée. Les artistes sont affichés 10 par 10. Et la mise à jour du graphique se fait dynamiquement, sans rechargement de la page.
- Il y a plusieurs traitements en JS faits au préalable. Tout d'abord, remplissage d'un tableau associatif avec le nombre de chansons explicites, non explicites et inconnues par id d'artiste. Ensuite, on transforme ce tableau en un tableau indexé, afin de pouvoir le trier. Le tri se fait de l'artiste ayant composé le plus de chansons, à celui qui en a composé le moins.

Pour générer le graphique, nous nous sommes aidés de la librairie ChartJS. Elle est simple d'intégration et très performante. Ce qui permet de rafraîchir le graphique sans quasiment aucune latence.

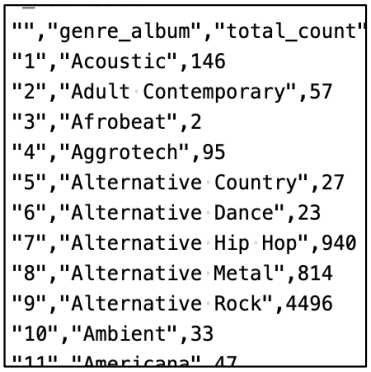
Partie de Kévin Constantin

Sujet de visualisation : *Les genres musicaux des musiques contenant des insultes*
Technique de visualisation : *Zoomable Sunburst*

Le Sunburst fonctionne sur un principe de hiérarchie, une classe, des sous-classes et des éléments finaux. Le but va être en premier lieu de construire cette hiérarchie sur les genres musicaux. A partir de la hiérarchie, on va construire un fichier au format JSON, utilisé par le site [1] sur lequel la visualisation a été réalisé.

Construction de la hiérarchie

Pour établir la hiérarchie sur les genres musicaux, il faut d'abord récupérer ces genres. J'ai créé un script[2] en R qui regroupe l'ensemble des musiques (ayant un genre et contenant des insultes) sur le genre et les compte.



```
"","genre_album","total_count"
"1","Acoustic",146
"2","Adult Contemporary",57
"3","Afrobeat",2
"4","Aggrotech",95
"5","Alternative Country",27
"6","Alternative Dance",23
"7","Alternative Hip Hop",940
"8","Alternative Metal",814
"9","Alternative Rock",4496
"10","Ambient",33
"11","Americana",47
```

Figure 1- genre_counted.csv

Ce résultat[3] a été enregistré au format csv. Avec un script[4] en python j'ai récupéré dans une liste l'ensemble des éléments de la colonne « genre_album », et pour chacun j'ai rangé dans des dictionnaires de genres plus globaux, par exemple « deathcore » a été rangé dans le style « heavy_metal_style » dans le genre « rock ».

J'ai volontairement choisi une hiérarchie s'arrêtant sur 3 niveaux (le genre global, un ou plusieurs styles dans le genre global en question, et pour chaque style, une liste des genres parmi ceux présents dans la colonne « genre_album ».

```
rock = {'heavy_metal_style':['deathcore', 'goregrind',
        'rock_style':['lo-fi', 'shoegazing', 'rio', 'jc
        'funk_style':['g-funk', 'funk', 'funk metal',
        'punk_style':['emo', 'screamo', 'gothic', 'powe
hiphop = {'crunk_style':['crunk', 'crunkcore', 'trap'],
        'hip_hop_style':['alternative hip hop', 'frenc
jazz = {'jazz_style':['jazz', 'jazz fusion', 'vocal jaz
        'blues_style':['blues', 'country blues', 'doo-v
country = {'country_style':['country', 'sertanejo', 'am
folk = {'folk_style':['filk', 'anti-folk', 'contempora
reggae = {'reggae_style':['reggae', 'reggae fusion', 'r
samba = {'samba_style':['samba', 'bossa nova', 'mpb']}
```

Figure 2- Extrait de l'ensemble des dictionnaires réalisés

[1] https://observablehq.com
[2] Nommé « script_making_genrecountedcsv.R » sur le Github du projet
[3] Nommé « genre_counted.csv » sur le Github du projet
[4] Nommé « script_genre_python.py » sur le Github du projet

Une fois l'ensemble des genres triés dans des dictionnaires, il faut rassembler ces dictionnaires en un unique, appelé ci-dessous « arbre ».

```
tree = {'unknown':unknown, 'electronic':electronic, 'rock':rock, 'hiphop':hiphop,
'jazz':jazz, 'country':country, 'folk':folk, 'reggae':reggae, 'samba':samba, 'rnbsoul':rnbsoul, 'pop':pop, 'rap':rap}
```

Figure 3- Arbre regroupant l'ensemble des genres globaux

La hiérarchie étant désormais établie, on peut passer à la création du JSON qui doit contenir les données de comptage sur les différents genres.

Construction du JSON

Dans le fichier de données utilisé dans l'exemple[5] sur le site où j'ai fait la visualisation, j'ai observé la manière dont le JSON était fait, avec le rendu visuel pour pouvoir refaire de-même avec des données différentes.


```
{
 "name": "flare",
 "children": [
  {
   "name": "analytics",
   "children": [
    {
     "name": "cluster",
     "children": [
      {"name": "AgglomerativeCluster", "value": 3938},
      {"name": "CommunityStructure", "value": 3812},
      {"name": "HierarchicalCluster", "value": 6714},
      {"name": "MergeEdge", "value": 743}
     ]
```


Figure 4- Extrait des données JSON de l'exemple du Zoomable Sunburst

Dans mon utilisation, « flare » qui est la plus haute hiérarchie, serait l'arbre. « analytics » serait un genre global, par exemple « rock ». « cluster » serait un style, par exemple « hard_metal_style ». L'ensemble des enfants de cet objet comme « AgglomerativeCluster » serait un genre propre au hard metal, par exemple « deathcore » avec en valeur le nombre de musique ayant pour genre « deathcore ». Cette information est déjà disponible dans les données du fichier *genre_counted.csv.*

Pour faciliter l'accès à cette dernière information, j'ai converti le fichier en un dictionnaire avec pour clé le **nom du genre** et en valeur **le comptage**. Il a ensuite « simplement » suffit d'itérer sur les éléments présents dans l'arbre en descendant dans la hiérarchie pour ensuite aller chercher grâce au dictionnaire la valeur du comptage. En pseudo-algorithme ça donne ceci : *pour chaque genre global dans l'arbre => pour chaque style dans le genre global => pour chaque genre dans le style => récupérer valeur avec la clé genre.*

La création du JSON avec les différents éléments décrit, a été réalisé en python[6].

[5] https://observablehq.com/@d3/zoomable-sunburst
[6] Nommé « script_json_python.py » sur le Github du projet

La visualisation

Ci-dessous, le résultat[7] avec les données extraites (sur environ 84k musiques)

Figure 5- Visualisation état de base

La visualisation affiche 2 niveaux de hiérarchie, les genres et les styles associés. Il est possible de cliquer un genre global (cercle intérieur) ou un style (cercle extérieur) pour déclencher un zoom sémantique, et avoir accès à plus d'informations : les genres.

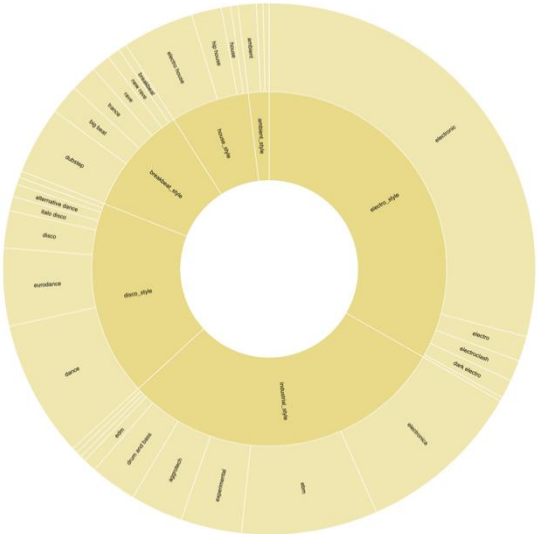


Figure 6- Visualisation après clique sur le genre global "electronic"

Dans la figure ci-contre, on peut désormais observer les différents genres présents dans les différents styles. Il est possible de cliquer sur un style pour afficher uniquement les genres associés.

Pour dézoomer, il suffit de cliquer au centre pour remonter dans la hiérarchie.

Pour visualiser le nombre de musique sur un genre global/style/genre donné, il suffit de laisser la souris sur l'éléments en question pour qu'une petite fenêtre s'affiche avec le nombre correspondant.

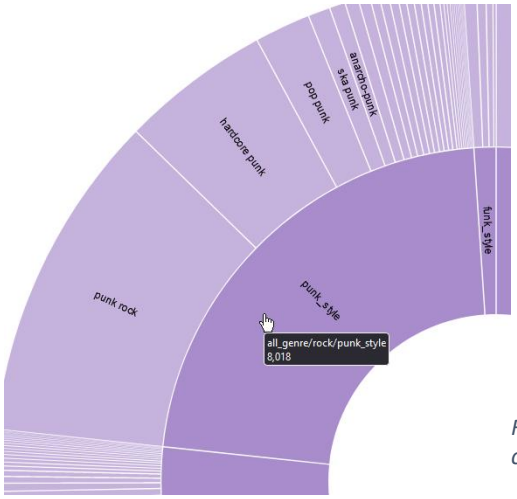


Figure 7- Visualisation du nombre de musique contenant des insultes appartenant au style punk

[7] Accessible sur cette adresse : https://observablehq.com/d/dcef123c394a46b4@358