C++ et Statistiques

Constantin Keuky

27 août 2025

Résumé

Ce projet a pour objectif de coder en C++ les outils statistiques élémentaires nécessaires à l'étude d'un dataset sur les performances Spotify. Une spécificité de ce projet est l'absence d'utilisation de bibliothèque pour effectuer les calculs statistiques.

0.1 Description des classes

0.1.1 Forme générale

Il existe 9 classes dans ce projet. 5 sous forme .cpp et 4 sous forme .h. La raison étant que le main n'a pas besoin de .h car il appelle les autres classes, autrement chaque classe qui n'est pas le main a son fichier .h associé.

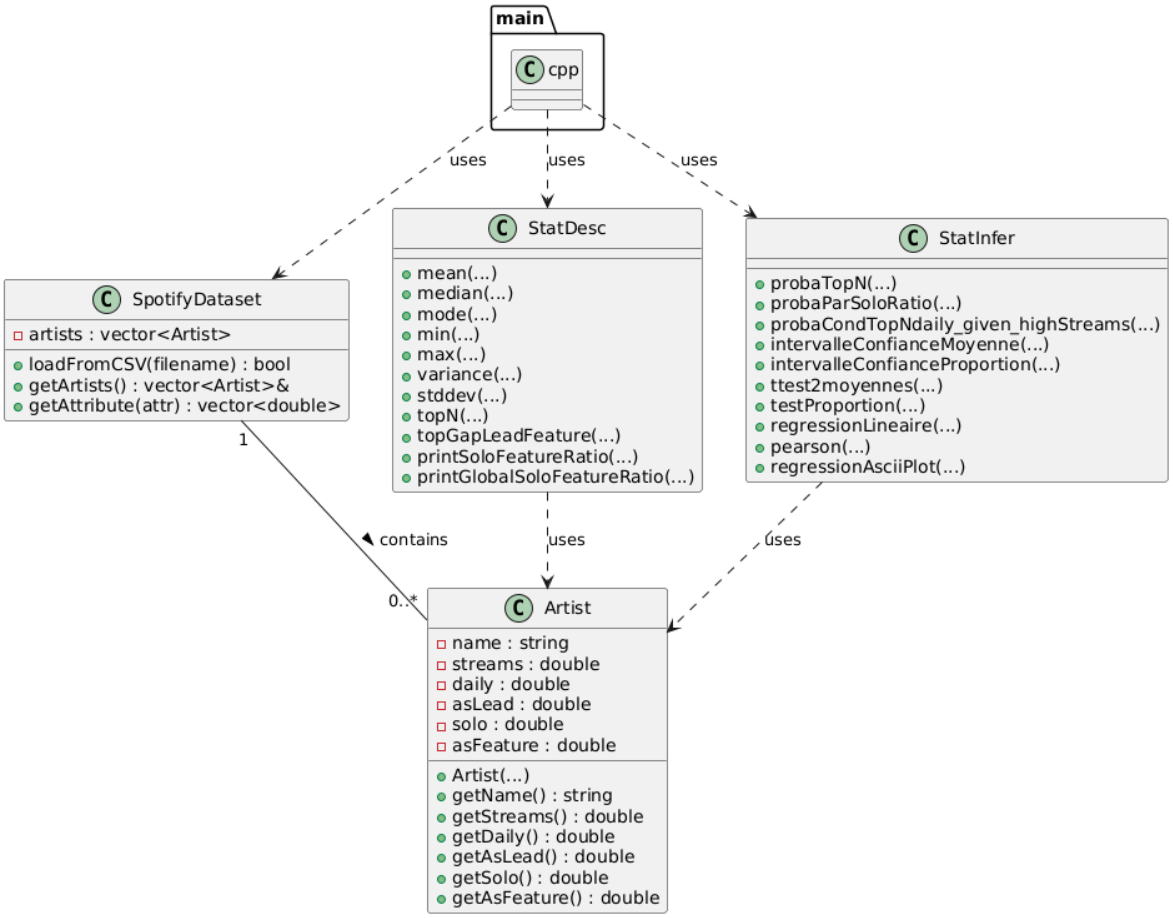

FIGURE 1 – Schéma UML du projet

Il y a donc :
— Artist.cpp/h, qui est notre format d'étude pour ce projet
— SpotifyDataset.cpp/.h, la classe qui permet l'extraction du fichier .csv
— StatDesc.cpp/.h, la classe qui contient les méthodes de statistiques descriptives
— StatInfer.cpp/.h, la classe qui contient les méthodes de statistiques inférentielles plus quelques probabilités bayésiennes
— main.cpp, le main pour exécuter tout ça

On compte en plus dans le dossier src le fichier .csv du dataset "Artists.csv", le fichier "main.exe" qui initialise l'application et le fichier "sauve" qui est une sauvegarde manuelle des calculs effectués.

Extrait de artists.csv					
Artist	Streams	Daily	As lead	Solo	As feature
Billie Eilish	"29,173.3"	19.313	"29,173.3"	"25,240.5"	NULL
XXXTENTACION	"28,370.7"	11.181	"24,681.3"	"20,244.5"	"3,689.4"
Chris Brown	"28,314.1"	12.788	"14,904.7"	"7,064.1"	"13,409.3"
Imagine Dragons	"28,100.5"	16.133	"26,430.9"	"22,543.8"	"1,669.6"
Khalid	"27,776.0"	8.833	"14,779.4"	"9,759.3"	"12,996.6"

0.1.2 Artist

.h

"pragma once", pour éviter les multi inclusions sachant que cette classe va être la plus utilisée "include <string>", pour avoir accès aux chaines de caractère
Dans le "private :" : la déclaration de tous les attributs qui sont ceux que l'on retrouve dans le fichier .csv
Dans le "public :" : en premier la déclaration de la méthode constructeur qui a tous les paramètres, puis les getters

.cpp

"include "Artist.h"", son fichier header dont il va définir le contenu des méthodes déclarées.
Puis le contenu des méthodes précédentes, majoritairement de la lecture de données

0.1.3 SpotifyDataset

.h

"include "Artist.h"", pour utiliser les méthodes de Artist qui sont necessaires à l'extraction ordonnée du fichier csv
"include <vector>", pour utiliser les vecteurs, ce choix permet de simplifier le developpement les arrays étant un peu plus complexe à manipuler
Dans le "private" :

— trim, normalizeKey et parseCSVLine qui sont des méthodes helpers pour le travail d'extraction
— ColMap, une structure qui reprend la forme attendue de nos données propres
— builColumnMap qui définit le nom des colonnes à partir de la première ligne
— parseNumber, transforme les données du csv en type double

Dans le "public" :

— loadFromCSV qui extrait les valeurs du fichiers
— getArtists qui convertit les données sous format artiste
— getAttribute qui est un getter pour les propriétés des artistes

.cpp

"include "SpotifyDataset.h"", son fichier header dont il va définir le contenu des méthodes déclarées
"include <fstream>", bibliothèque pour manipuler les fichiers .txt
"include <sstream>", pour plus facilement manipuler les string
"include <cctype>", pour avoir accès à des méthodes de manipulation des caractères individuels
Puis le contenu des méthodes précédentes, majoritairement de l'extraction et formatage de données

0.1.4 StatDesc

.h

"include "Artist.h"", car c'est notre format de travail

Dans le "public" :

— mean qui fait la moyenne
— median qui trouve la médianne
— mode qui trouve la valeur mode du dataset
— min et max pour les valeurs minimales et maximales
— variance qui fait la variance d'une variable
— stddev pour standart deviation-l'écart type
— topN qui donne les premiers N max d'un attribut
— topGapLead qui donne l'écart max au sein d'une variable
— printSoloFeatureRatio qui fait la comparaison entre les apparitions solo ou feat des artistes
— printGlobalSoloFeatureRatio qui fait ce calcul pour tous les artistes

.cpp

"include <map>", pour pouvoir utiliser les clefs
"include <cmath>", qui donne accès à quelques fonctions mathématiques de référence comme exp-log-sin... Ne donne pas accès à des méthodes explicites de calcul statistique
Puis le contenu des méthodes précédentes, majoritairement des statistiques descriptives

0.1.5 StatInfer

.h

"include "Artist.h"", car c'est notre format de travail
Dans le "public" :

- probaTopN la probabilité de tirer au hasard un artiste faisant partie du top n configurable selon les streams
- probaParSoloRatio la probabilité qu'un artiste ait plus de n%, aussi configurable, de ses streams en solo
- probaCondTopNdailygivenhighStreams la proba d'être top n configurable daily si streams > seuilStreams
- intervalleConfianceMoyenne qui fait l'IC 95% sur la moyenne
- intervalleConfianceProportion qui fait l'IC sur une proportion par demi-largeur
- ttest2moyennes le test de student sur 2 moyennes
- testProportion via z-test
- regressionLineaire de forme simple aX + b
- pearson qui calcule le coefficient éponyme
- regressionAsciiPlot qui affiche graphiquement le résultat de la régression linéaire

.cpp

"include <cmath>", qui donne accès à quelques fonctions mathématiques de référence comme exp-log-sin... Ne donne pas accès à des méthodes explicites de calcul statistique
Puis le contenu des méthodes précédentes, majoritairement des statistiques inférentielles

0.1.6 Main

"include "SpotifyDataset.h"", appel de la classe d'extraction, par ailleurs pas besoin de rappeler la classe Artist étant déjà appelée dans cette classe
"include "StatDesc.h"", appel de la classe de statistiques descriptives
"include "StatInfer.h"", appel de la classe de statistiques inférentielles
define COLORGREEN "033[1 ;32m" define COLORRESET "033[0m" define COLOR-BOLD "033[1m", pour mettre la console en vert IBM.
lastResult, qui sert à stocker les lignes pour la sauvegarde, il est d'accessibilité globale.
Et les méthodes :

- handleXCommand, ce sont des méthodes qui permettent de simplifier l'appel des méthodes de calcul statistique dans le main. Toutes les méthodes ne sont pas sous ce format, certaines sont explicitement appelées et configurées directement dans le main. Les méthodes handleXCommand existent surtout par soucis de clarté
- split et saveToFile qui permettent la segmentation et la sauvegarde des résultats dans un fichier distinct
- showMenu qui donne un affichage complet des commandes disponibles avec exemples
- main qui appelle toutes les méthodes précédentes, il est bouclé pour une exécution continue sauf si on le quitte.

0.2 Traitement du CSV

0.2.1 Spécifications

Les classes SpotifyDataset chargent un fichier CSV d'artistes Spotify, nettoient et convertissent les champs, puis construisent un objet Artist par ligne valide. Elles ont les caractéristiques suivantes :

- respecter le format CSV standard (RFC 4180) : guillemets, virgules internes, guillemets échappés
- tolérer des variations d'en-têtes (ordre et alias de colonnes)
- gérer des conventions numériques courantes (séparateurs de milliers, virgule décimale)
- rester résilient : une ligne invalide n'interrompt pas l'import ; un bilan est journalisé

Le format attendu est de 6 colonnes :

1. Artist (texte)
2. Streams (numérique)
3. Daily (numérique)
4. As lead (numérique)
5. Solo (numérique)
6. As feature (numérique)

Ces colonnes peuvent être réordonnées et renommées si la première ligne est un en-tête car elles y sont dérivées.

0.2.2 Méthodes

API externe

Les méthodes suivantes sont accessibles par toutes les classes qui font appel à SpotifyDataset.cpp :

- bool loadFromCSV(const std : :string& filename) lit et parse le fichier, remplit un std : :vector<Artist>. retourne false si le fichier ne peut pas être ouvert ; true sinon (même si des lignes sont ignorées).
- const std : :vector<Artist>& getArtists() const renvoie la liste des artistes importés.
- std : :vector<double> getAttribute(const std : :string& attr) const extrait un vecteur numérique pour un attribut (streams, daily, solo, aslead/as_lead, asfeature/as_feature).

API interne

- parseCSVLine : découpe une ligne en champs, compatible RFC 4180 (guillemets, virgules internes, guillemets échappés).
- buildColumnMap : détecte la présence d'un en-tête et mappe les libellés vers les indices de colonnes.

— parseNumber : conversion texte → double avec gestion des groupements et de la virgule décimale.
— trim / normalizeKey : utilitaires de nettoyage et de normalisation.

0.2.3 Pipeline de traitement

Ouverture du fichier

Ouverture via std : :ifstream.
Si échec : loadFromCSV renvoie false (aucune donnée chargée).

Première ligne : en-tête ou données ?

La première ligne est lue et découpée via parseCSVLine. Heuristique d'en-tête avec buildColumnMap : si au moins 3 libellés reconnus (artist/streams/etc.), la ligne est traitée comme un en-tête. Deux cas :
— En-tête reconnu : on retient un mapping colonne→indice basé sur les libellés.
— Pas d'en-tête : mappage positionnel fixe (0..5) et la première ligne est traitée comme des données (pas de perte de la première data row).

Lecture des lignes suivantes

Pour chaque ligne :

1. Découpage robuste via parseCSVLine (gère guillemets, virgules internes, guillemets échappés).
2. Filtrage des lignes vides.
3. Validation simple du nombre de colonnes.
4. Conversion des champs numériques via parseNumber.
5. Si toutes les conversions réussissent : construction de Artist(name, streams, daily, asLead, solo, asFeature) et insertion dans le vecteur.
6. En cas d'erreur (conversion impossible, nom vide, etc.) : la ligne est ignorée ; un message explicite est journalisé sur std : :cerr.

0.2.4 Détails sur le Parsing CSV (RFC 4180)

Gestion des guillemets

Les champs peuvent être entourés de guillemets pour contenir des virgules littérales. Les guillemets internes sont échappés en les doublant : "" représente ".
parseCSVLine parcourt la ligne caractère par caractère avec un état inQuotes :
— " inverse l'état, sauf si c'est un guillemet échappé ("" → ajoute un " au champ).
— une virgule hors guillemets sépare les champs.
— fin de ligne → on pousse le champ courant.
Chaque champ est nettoyé (trim) en entrée et en sortie de parseCSVLine.

Exemples de découpage

Entrée : Taylor Swift,"85,041.3",412.6,"70,000",60000,25041.3
Sortie : ["Taylor Swift", "85,041.3", "412.6", "70,000", "60000", "25041.3"]
Entrée : "AC/DC","He said ""Hello"""
Sortie : ["AC/DC", "He said "Hello""]

Détection d'en-tête et mapping des colonnes

normalizeKey supprime espaces/underscores/tirets et met en minuscules ;
ex. : "As lead", as_lead, AS-LEAD → aslead.
buildColumnMap reconnaît notamment :

- artist, name → nom d'artiste
- streams, stream → total de streams
- daily → métrique journalière
- aslead, lead, asprincipal → streams en lead
- solo → streams solo
- asfeature, feature, feat → streams en feature

Si la première ligne est un en-tête (plus de 3 noms reconnus), on utilise ce mapping.
Sinon, on utilise le mapping positionnel 0..5 et on traite la ligne 1 comme des données.

Étapes de parseNumber :

Trim du champ.
Chaîne vide → retourne 0.0 (politique par défaut ; peut être changée pour « vide = erreur »).
Suppression des espaces internes typiques de groupement ("60 000" → "60000").
Gestion virgule/point :
s'il y a des virgules et pas de point :
si une seule virgule : interprétée comme virgule décimale ("85,3" → "85.3") ;
si plusieurs virgules : considérées comme séparateurs de milliers (toutes supprimées : "85,041,300" → "85041300").
s'il y a virgules et point : les virgules sont des séparateurs de milliers (supprimées).
sinon : entier « pur ».
Conversion par std : :stod (accepte +/-, notation exponentielle, etc.).
Si des caractères traînants subsistent après la conversion, un avertissement est émis.
En cas d'échec : message d'erreur contextualisé et exception relancée (la ligne sera ignorée par le code appelant).

Exemples de conversion

"85,041.3" → 85041.3 (milliers + point décimal)
"85,3" → 85.3 (virgule décimale)
"60 000" → 60000
"" → 0.0 (champ vide)

Remarque sur l'ambiguïté du style "1,234"

Le choix par défaut est "une seule virgule sans point = virgule décimale" (1,234 → 1.234). Si à la place on a la virgule comme séparateur de milliers (1,234 → 1234), il est possible d'ajuster ce comportement.

0.2.5 Gestion des erreurs et résilience

Granularité : l'erreur sur un champ d'une ligne fait ignorer la ligne entière, mais l'import continue.
Journalisation : erreurs/avertissements sur std : :cerr avec numéro de ligne et valeur incriminée.
Bilan final : message "X ligne(s) importée(s), Y ignorée(s). Total artistes : Z".
Sémantique de retour : loadFromCSV renvoie true si le fichier s'ouvre, même si certaines lignes sont ignorées.
Idempotence : artists.clear() est appelé en début de loadFromCSV pour remplacer un import précédent.

0.2.6 Complexité et performance

Temps : linéaire dans la taille du fichier. parseCSVLine et std : :stod dominent le coût. Mémoire : un Artist par ligne valide ; pas de préallocation (possible d'ajouter reserve si on connaît la taille).

0.2.7 Cas limites

Retours à la ligne à l'intérieur d'un champ guillemeté : non supportés (nécessiterait un lecteur multi-lignes).
Unicode/locale : la normalisation des en-têtes repose sur isalnum/isspace de la locale C ; des en-têtes très exotiques peuvent ne pas être reconnus.
Ambiguïté virgule décimale vs séparateur de milliers : la règle choisie fonctionne avec notre modèle mais n'est pas universel.

0.2.8 Exemples d'extraction de ligne

Cas 1 — guillemets + virgules internes Taylor Swift,"85,041.3",412.6,"70,000",60000,25041.3 → découpage correct ; conversions 85,041.3 → 85041.3, "70,000" → 70000 ; ligne importée.
Cas 2 — décimale européenne Amelie,"85,3",410,70000,60000,25341 → "85,3" → 85.3 ; ligne importée.
Cas 3 — champ numérique vide Unknown Artist,"",0,"","","" → champs vides → 0.0 ; ligne importée (politique par défaut).

0.2.9 Améliorations possibles

Rapport d'import structuré : ajouter une surcharge loadFromCSV(..., LoadReport&) pour retrouver dans le code le nombre exact de lignes importées/ignorées et les détails d'erreurs.

Gestion du cas “valeur vide” : rendre configurable le choix “vide = 0.0” vs “vide = erreur”.
Choix virgule décimale vs milliers : rendre cette règle configurable au niveau du dataset.
Préallocation de mémoire : si les volumes sont grands et la taille connue.

0.3 Méthodes de calcul

0.3.1 Moyenne

$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$

Mesure de tendance centrale qui résume une distribution par la valeur “moyenne” des observations. Très sensible aux valeurs extrêmes ; utile quand les données sont relativement symétriques ou quand on souhaite une mesure globale du niveau. Desc mean streams donne : Moyenne de streams : 2914.44
Desc mean daily donne : Moyenne de daily : 1.85081

0.3.2 Médiane

$$\tilde{x} = \begin{cases} x_{(\frac{n+1}{2})} & \text{si } n \text{ est impair} \\ \dfrac{x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}}{2} & \text{si } n \text{ est pair} \end{cases} \qquad (\text{sur } x_{(1)} \leq \cdots \leq x_{(n)})$$

Valeur centrale d’une distribution une fois les données triées. Plus robuste aux valeurs extrêmes que la moyenne, préférable en présence d’asymétrie ou d’outliers. Comme dans notre cas.

Desc median streams donne : Mediane de streams : 1459.4
Desc median daily donne : Mediane de daily : 0.8695

0.3.3 Mode(s)

Mode $= argmax_v |i : x_i = v|$

Valeur(s) la ou les plus fréquente(s) dans l’échantillon. Sur des données continues (réelles), le mode exact est souvent peu informatif car il y a beaucoup de données uniques. Ici nous avons tout de même quelques valeurs qui se répètent.

Desc mode streams donne : Mode(s) de streams : 731.6
Desc mode daily donne : Mode(s) de daily : 0.405 0.52 0.587

0.3.4 Minimum / Maximum

$\min(x) = \min_{1 \leq i \leq n} x_i \quad , \quad \max(x) = \max_{1 \leq i \leq n} x_i$

Les bornes observées de la distribution. Utile pour détecter des valeurs anormales ; entièrement déterminé par les extrêmes (très sensible aux outliers).

Desc min streams donne : Minimum de streams : 728.6
Desc min daily donne : Minimum de daily : 0
Desc max streams donne : Maximum de streams : 85041.3
Desc max daily donne : Maximum de daily : 85.793

0.3.5 Amplitude

$\text{amp}(x) = \max(x) - \min(x)$

Différence entre la valeur maximale et la valeur minimale. Mesure simple de dispersion, très sensible aux extrêmes.

Desc amplitude streams donne : Amplitude de streams : 84312.7
Desc amplitude daily donne : Amplitude de daily : 85.793

0.3.6 Variance (population et échantillon)

$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \quad , \quad s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$

Mesure la dispersion autour de la moyenne. Par la suite on utilisera la version échantillon.

Desc variance streams donne : Variance de streams : 2.32932e+07
Desc variance daily donne : Variance de daily : 14.1033

0.3.7 Écart-type

$\sigma = \sqrt{\sigma^2} \quad , \quad s = \sqrt{s^2}$

Racine carrée de la variance ; s'exprime dans la même unité que la variable. Interprétation directe comme "dispersion moyenne" autour de la moyenne.

Desc stddev streams donne : Ecart-type de streams : 4826.31
Desc stddev daily donne : Ecart-type de daily : 3.75544

0.3.8 Top-N (tri décroissant d'un attribut a)

Soit π une permutation telle que $a_{\pi(1)} \geq \cdots \geq a_{\pi(n)}$, $\quad$ Top-$N = \pi(1), \ldots, \pi(N)$
Classement par ordre décroissant d'une métrique (ex. streams, daily, solo) pour extraire les N premières entrées. Met en évidence les "leaders" selon l'attribut choisi ; l'égalité de valeurs n'est pas spécifiquement traitée (ordre d'origine en cas d'égalité).

top 10 streams donne : Top 10 artistes selon streams :
1. Drake (streams = 85041.3)
2. Bad Bunny (streams = 67533)
3. Taylor Swift (streams = 57859)
4. The Weeknd (streams = 53665.2)
5. Ed Sheeran (streams = 47907.7)
6. Justin Bieber (streams = 47525.7)
7. Eminem (streams = 42029)
8. Ariana Grande (streams = 40111)
9. J Balvin (streams = 38774.8)
10. Post Malone (streams = 38002.7)

top 10 solo donne : Top 10 artistes selon solo :
1. Taylor Swift (solo = 50425.7)
2. Ed Sheeran (solo = 33917)
3. Drake (solo = 32681.6)
4. The Weeknd (solo = 31164.2)
5. BTS (solo = 28991.6)
6. Billie Eilish (solo = 25240.5)
7. Ariana Grande (solo = 23307.3)
8. Bad Bunny (solo = 23073)
9. Imagine Dragons (solo = 22543.8)
10. Coldplay (solo = 22329.9)

top 10 aslead donne :
Top 10 artistes selon aslead :
1. Drake (aslead = 57252.6)
2. Taylor Swift (aslead = 55566.7)
3. Ed Sheeran (aslead = 42767.9)
4. The Weeknd (aslead = 42673.3)
5. Bad Bunny (aslead = 40969.6)
6. Eminem (aslead = 35475.8)
7. Post Malone (aslead = 34494)
8. Ariana Grande (aslead = 33219.8)
9. BTS (aslead = 32041.3)
10. Billie Eilish (aslead = 29173.3)

0.3.9 Plus grand écart —asLead ou asFeature—

$\Delta_i = \big|, \text{asLead}_i - \text{asFeature}_i, \big|$; trier Δ_i décroissant et prendre les N premiers

Mesure la "spécialisation" d'un artiste entre ses streams en tant que principal et en tant qu'invité. Un écart élevé indique un profil très "lead" ou très "feature" ; le classement par écart absolu met ces profils en avant.

top gapleadfeature 5 donne : Top 5 ecart —asLead - asFeature— :
1. Taylor Swift (asLead=55566.7, asFeature=2292.4, ecart=53274.3)
2. Ed Sheeran (asLead=42767.9, asFeature=5139.8, ecart=37628.1)
3. The Weeknd (asLead=42673.3, asFeature=10991.9, ecart=31681.4)
4. Post Malone (asLead=34494, asFeature=3508.6, ecart=30985.4)
5. Drake (asLead=57252.6, asFeature=27788.7, ecart=29463.9)

0.3.10 Ratios par artiste (sur le total de l'artiste)

$p_i^{(\text{solo})} = \frac{\text{solo}_i}{\text{streams}_i}$, $p_i^{(\text{feat})} = \frac{\text{asFeature}_i}{\text{streams}_i}$

Part des streams solo et part des streams en feature rapportées au total de l'artiste. Permet de comparer la structure des streams indépendamment du volume global ; nécessite un total non nul.

repartition donne :
Artiste %solo %feature

Drake 38.43 32.68

Bad Bunny 34.17 39.33
Taylor Swift 87.15 3.96
The Weeknd 58.07 20.48
Ed Sheeran 70.80 10.73
Justin Bieber 36.16 41.11
Eminem 51.34 15.59
Ariana Grande 58.11 17.18
J Balvin 14.70 54.99
Post Malone 49.85 9.23
Kanye West 47.87 27.77
Travis Scott 39.91 49.75
...

0.3.11 Répartition globale (sur la somme du dataset)

$p^{(\text{solo})}\text{glob} = \frac{\sum i=1^n \text{solo}i}{\sum i=1^n \text{streams}i}$, $p^{(\text{feat})}\text{glob} = \frac{\sum_{i=1}^n \text{asFeature}i}{\sum i=1^n \text{streams}i}$, $p^{(\text{autre})}\text{glob} = 1 - p^{(\text{solo})}\text{glob} - p^{(\text{feat})}\text{glob}$

Parts agrégées des streams solo et feature sur l'ensemble du jeu de données ; la part "autre" est le complément. Vue d'ensemble des poids relatifs ; utile pour contextualiser les ratios individuels.

repartition global donne : Répartition globale des streams :
% solo : 50.95
% feature : 26.53
Autre : 22.52

0.3.12 Proba uniforme d'être dans le Top-N (attr)

$P = \frac{N}{n}$

Modèle simple où chaque artiste a la même chance d'appartenir au Top-N. Ne dépend pas des données, sert de baseline.

proba top 10 streams donne : Proba d'etre dans le top 10 de streams (modele uniforme n/N) : 0.00333333

0.3.13 Proba qu'un artiste ait un ratio solo ¿ seuil r

$P = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}! \left(\frac{\text{solo}_i}{\text{streams}_i} > r \right)$

Estimation empirique par fréquence d'artistes dont le ratio solo dépasse le seuil choisi. Sensible au traitement des cas où le total de streams est nul.

Proba solo70 donne : Proba qu'un artiste ait >70% de streams solo : 0.399

0.3.14 Proba conditionnelle : être dans le Top-N daily parmi streams > s

$T = \text{Top-}N$ par daily, $A = i : streams_i > s$, $P = \frac{(A \cap T)}{A}$

On identifie d'abord le Top-N global selon "daily". On conditionne ensuite à l'ensemble des artistes dont les streams dépassent un seuil, puis on calcule la proportion

appartenant aussi au Top-N global. Interprétation : “Parmi les artistes au-dessus du seuil, quelle fraction figure dans le Top-N daily global ?”

proba condtop10daily 10000 donne : Proba(d'etre dans le top10 daily GLOBAL — streams > 10000) = 0.0551724

0.3.15 Intervalle de confiance (moyenne, approx. normale)

$\bar{x} \pm z_{1-\alpha/2}, \frac{s}{\sqrt{n}}$ (ici $z_{0.975} \approx 1.96$)

Intervalle symétrique autour de la moyenne échantillonnée, dont la demi-largeur dépend de l'écart type et de la taille d'échantillon. Utile si la taille est suffisante ou si la distribution des moyennes est proche de la normale.

ic mean streams donne : IC 95% pour la moyenne de streams : [2741.73 ; 3087.15]

0.3.16 Intervalle de confiance (proportion, approx. normale)

$\hat{p} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$, $\hat{p} = \frac{\text{succès}}{n}$

Intervalle centré sur la proportion observée, avec une demi-largeur qui décroît avec la taille d'échantillon. Fonctionne si l'échantillon est suffisamment grand et la proportion pas trop proche de 0 ou 1.

ic prop streams 10000 donne : IC 95% pour la proportion d'artistes avec streams > 10000 : [0.0406586 ; 0.056008]

0.3.17 Test t (Welch) pour deux moyennes

$t = \frac{\bar{x}-\bar{y}}{\sqrt{\frac{s_x^2}{n_x}+\frac{s_y^2}{n_y}}}$

Compare deux moyennes en autorisant des variances inégales et des tailles d'échantillons différentes. Retourne la statistique t. L'interprétation se fait via des seuils usuels (par exemple, un t en valeur absolue au-delà d'environ 2 peut indiquer une différence significative à 5% selon les degrés de liberté).

test ttestsolofeature donne : T-statistique pour comparaison des moyennes (solo vs feature) : 11.8227 ($|t| >= 2$ => significatif a 5% environ)

0.3.18 Test de proportion ($z - test, H0 : p = p_0$)

$z = \frac{\hat{p}-p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$

Compare la proportion observée à une proportion théorique sous l'hypothèse nulle. Renvoie une statistique z ; interprétation via seuils usuels (environ 1,96 en valeur absolue pour 5%), sous condition d'approximation normale valable.

test testprop streams 1500 0.5 donne : Test de proportion (H0 : p = 0.5) : z = -1.7162 ($> 1.96 ou < -1.96$ = significatif á 5%)

0.3.19 Régression linéaire simple (Y = aX + b)

$\bar{x} = \frac{1}{n}\sum x_i, \quad \bar{y} = \frac{1}{n}\sum y_i$

$a = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$, $b = \bar{y} - a, \bar{x}$

$r = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\left(\sum_{i=1}^{n}(x_i-\bar{x})^2\right)\left(\sum_{i=1}^{n}(y_i-\bar{y})^2\right)}} \quad , \quad R^2 = r^2$

$e_i = y_i - (ax_i + b) \quad , \quad \bar{e} = \frac{1}{n}\sum e_i \quad , \quad s_e = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(e_i - \bar{e})^2}$

Modélise une relation linéaire entre une variable explicative X et une variable réponse Y. La pente indique la variation moyenne de Y pour une unité de X ; l'ordonnée à l'origine est la valeur de Y attendue quand X vaut zéro. Le coefficient de détermination R^2 mesure la part de variance de Y expliquée par X. Les résidus (observé moins ajusté) permettent de diagnostiquer la validité des hypothèses : linéarité, homoscédasticité, erreurs centrées, absence d'outliers influents.

regression streams solo donne : Regression streams $->$ solo
Y = 0.464382 * X + 131.53 ; $R\hat{2}$ = 0.68154
Residuals : mean=-4.56339e-13, std=1532.04, min=-6941.55, max=0

regression streams aslead donne : Regression streams $->$ aslead
Y = 0.700593 * X + 99.4731 ; $R\hat{2}$ = 0.889788
Residuals : mean=1.10087e-11, std=1190.01, min=-2426.2, max=0

regression aslead asfeature donne : Regression aslead $->$ asfeature
Y = 0.27005 * X + 194.871 ; $R\hat{2}$ = 0.267406
Residuals : mean=-2.78552e-13, std=1602.24, min=12132.7, max=29.2

0.3.20 Corrélation de Pearson

$\rho = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\left(\sum_{i=1}^{n}(x_i-\bar{x})^2\right)\left(\sum_{i=1}^{n}(y_i-\bar{y})^2\right)}}$

Mesure l'intensité et le sens de la relation linéaire entre deux variables. Valeurs proches de 1 ou de -1 indiquent une relation linéaire forte (positive ou négative) ; proche de 0 indique une faible relation linéaire. Sensible aux valeurs extrêmes ; ne capture pas des relations non linéaires et n'implique pas la causalité.

correlation solo asfeature donne : Correlation de Pearson entre solo et asfeature : 0.345153

0.4 Analyse des résultats

0.4.1 Statistiques descriptives

A partir des résultats sur les moyennes et médianes on peut déduire que le dataset contient beaucoup de petits artistes mais que les artistes qui ont le plus de réussite ont de gros nombres sur les streams. Effectivement la moyenne des streams est presque le double de la médiane, quelques artistes très populaires tirent la moyenne vers le haut. Aussi de manière surprenante il existe un mode unique pour streams, ce qui indique que 731.6 a été atteint au moins 2 fois.
Les informations qu'on retire des min/max et amplitudes confirment l'hypothèse faite avec les moyennes et médianes. C'est à dire que l'écart entre médiane et top artistes est plus grand que celui entre bottom artistes et la médiane. En effet le min est 2 fois plus petit que la médiane alors que le top est 56 fois plus grand.
La variance est difficilement interprétable étant le carré de l'unité étudiée.
L'écart type est de la bonne dimension, encore une fois il indique que le dataset a

beaucoup de dispersion. Comme on a une médiane à 1500, une moyenne à 3000 et un minimum à 728. Un écart type de 4800 montre qu'il y a beaucoup de dispersion à droite.
Le top des plus grands écarts lead feature donne trois types d'artistes. Ceux qui ont très peu de feature comme Taylor Swift et Post Malone, ceux qui ont eu quelques features populaires comme The Weeknd avec ses feat Daft Punk et ceux qui ont beaucoup de features comme Drake.
Ces analyses sont reprises par le ratios par artistes où l'on voit notamment Taylor Swift qui a 87% de solo et 4% de features contrairement à Drake avec 38.5% et 33% de solo et feature respectivement.
La répartition globale nous donne que la majorité des titres sont des solos.

0.4.2 Probabilités

La probabilité d'être dans un top 10 est de $\frac{1}{300}$, car il y a 3000 artistes.
Pour le seuil des $> 70\%$ on a 0.399. Il y a donc une certaine quantité d'artistes qui font principalement des titres solo.
Pour la probabilité conditionnelle, le résultat est assez faible car le Top 10 est très sélectif et 10k ne suffit pas à le garantir.

0.4.3 Statistiques inférentielles

Avec l'intervalle de confiance sur la moyenne, on a 95% de confiance que la moyenne vraie des streams se situe dans [2741.73 ; 3087.15]. Ce qui confirme notre hypothèse qu'il y a une asymétrie à droite et que les artistes à droite de la médiane ont largement plus de réussite que ceux à gauche.
L'intervalle de confiance sur la proportion montre que le nombre d'artistes très populaires (au dessus de 10000 streams mensuel) est faible, entre 4 et 5%.
Le test de moyenne entre solo et feature donne pour résultat 11.8. C'est une valeur assez haute qui indique que les 2 variables ne suivent pas la même tendance.
Pour le test de proportion j'ai pris l'hypothèse que parmi tous les artistes, ceux qui ont plus que 1500 streams mensuel représentent la moitié de la population. On trouve -1,72 comme résultat ce qui n'est pas dans la région critique. On ne rejette donc pas H0 au seuil de 5%.
Pour les régressions on remarque que les streams et les leads sont le plus proche d'une relation un pour un.
La corrélation de Pearson entre solo et feature donne 0.35. C'est une corrélation positive modeste.