

**M1 - ISD**

Rapport de projet ML

*Prédiction de popularité des chansons*

---

**Sarah IKHLEF**

## Sommaire

<b>1. Objectif du projet</b>	<b>3</b>
<b>2. Données</b>	<b>3</b>
<b>3. Variables d'entrée</b>	<b>3</b>
<b>4. Variable de sortie</b>	<b>3</b>
<b>5. Techniques</b>	<b>3</b>
<b>6. Bilan de l'utilisation de ChatGPT</b>	<b>4</b>
<b>7. Annexes: Conversation avec ChatGPT</b>	<b>5</b>

# 1. Objectif du projet

---

Ce projet vise à développer un modèle de prédiction de “popularité” des chansons Spotify en utilisant des techniques d'apprentissage supervisé (Régression, Decision Tree, Random Forests).

## 2. Données

---

Les données utilisées sont récupérées sur Kaggle sous format CSV :

<https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset>

Le dataset comporte 20 features (colonnes) et 114000 lignes, chaque ligne représente une musique.

## 3. Variables d'entrée

---

Les différentes caractéristiques détaillées dans le lien ci-dessus (Artistes, album, durée de la chanson, énergie, ... etc).

## 4. Variable de sortie

---

Attribut “popularité” de la chanson qui prend ses valeurs entre 0 et 100.

## 5. Techniques

---

Ce projet est réalisé à l'aide de ChatGPT:

- Idée du projet
- Spécifications
- Étapes à suivre
- Code (à 90%) ... etc

## 6. Bilan de l'utilisation de ChatGPT

---

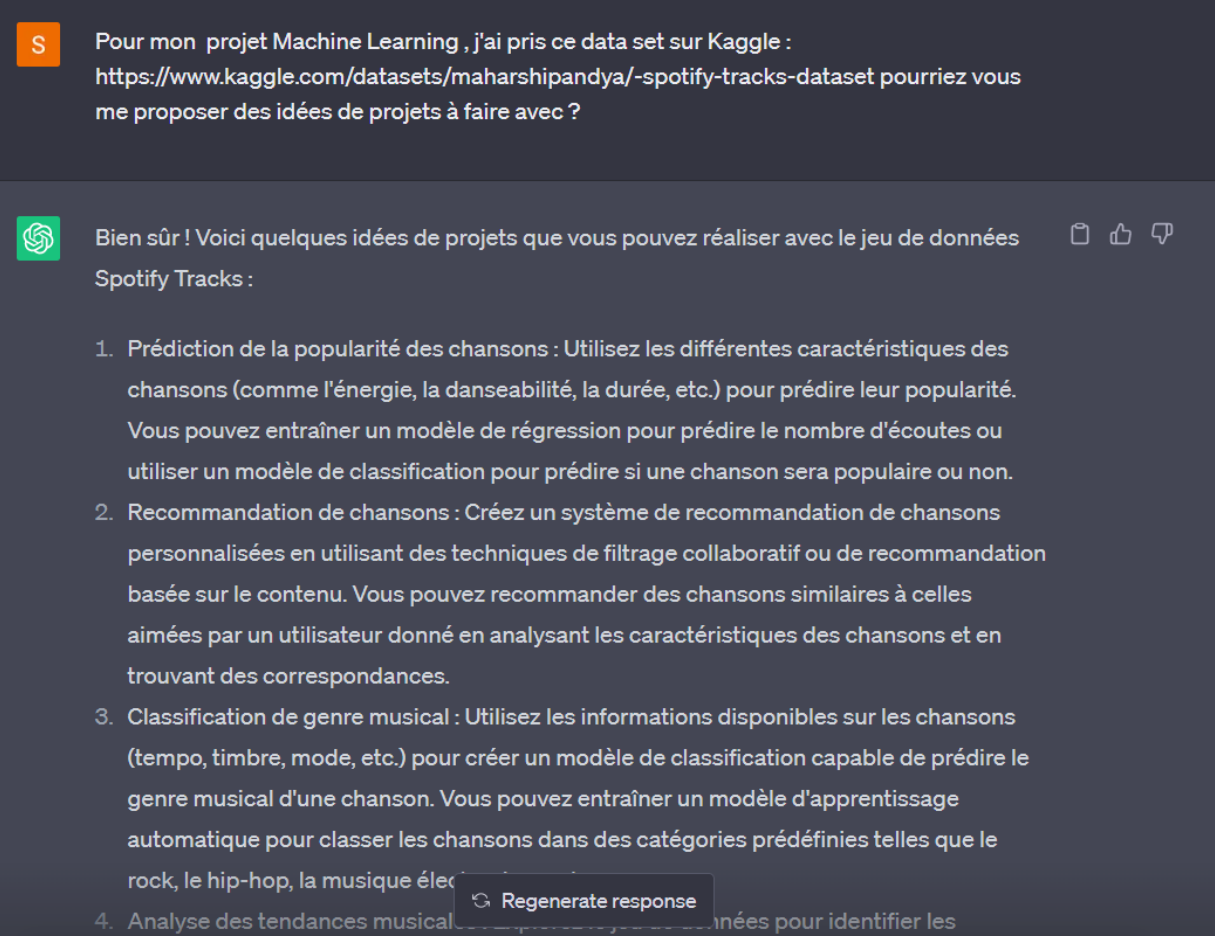
<u>Points positifs</u>	<u>Points négatifs</u>
<ul style="list-style-type: none"><li>- Il permet de générer du code qui répond au besoin de l'utilisateur avec les explications/commentaires nécessaires.</li><li>- ChatGPT m'a proposé plusieurs idées de projets seulement en lui fournissant le lien vers le dataset à utiliser.</li><li>- Après avoir choisi mon projet, ChatGPT m'a proposé les spécifications et les étapes à suivre afin de réaliser le projet.</li><li>- Il m'a guidé pour réaliser le projet de A à Z.</li><li>- Il donne des conclusions sur les résultats obtenus.</li><li>- Il fournit des explications aux problèmes rencontrés puis il suggère des solutions à envisager.</li></ul>	<ul style="list-style-type: none"><li>- Dans le cas de mon projet, la prédiction nécessite une exploration et une analyse de données. ChatGPT intervient pour fournir des observations, et suggérer les actions à effectuer sur les données, mais il ne prend pas de décision à la place de l'utilisateur.</li><li>- ChatGPT fournit parfois des réponses qui ne tiennent pas compte du contexte et donc il faut lui rappeler le contexte.</li></ul>

## 7. Annexes: Conversation avec ChatGPT

---

Voici quelques captures d'écran de la discussion avec ChatGPT concernant les grandes étapes du projet:

### 1) Choix du projet:



The screenshot shows a chat interface with a dark background. The user's message is in a light blue bubble, and the AI's response is in a dark blue bubble. The user asks for project ideas based on a Spotify dataset from Kaggle. The AI provides three detailed suggestions: predicting song popularity, recommending songs, and classifying musical genres. A 'Regenerate response' button is visible at the bottom of the AI's message.

**User:** Pour mon projet Machine Learning , j'ai pris ce data set sur Kaggle : <https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset> pourriez vous me proposer des idées de projets à faire avec ?

**ChatGPT:** Bien sûr ! Voici quelques idées de projets que vous pouvez réaliser avec le jeu de données Spotify Tracks :

1. **Prédiction de la popularité des chansons :** Utilisez les différentes caractéristiques des chansons (comme l'énergie, la danseabilité, la durée, etc.) pour prédire leur popularité. Vous pouvez entraîner un modèle de régression pour prédire le nombre d'écoutes ou utiliser un modèle de classification pour prédire si une chanson sera populaire ou non.
2. **Recommandation de chansons :** Créez un système de recommandation de chansons personnalisées en utilisant des techniques de filtrage collaboratif ou de recommandation basée sur le contenu. Vous pouvez recommander des chansons similaires à celles aimées par un utilisateur donné en analysant les caractéristiques des chansons et en trouvant des correspondances.
3. **Classification de genre musical :** Utilisez les informations disponibles sur les chansons (tempo, timbre, mode, etc.) pour créer un modèle de classification capable de prédire le genre musical d'une chanson. Vous pouvez entraîner un modèle d'apprentissage automatique pour classer les chansons dans des catégories prédéfinies telles que le rock, le hip-hop, la musique élec
4. **Analyse des tendances musicales**

Regenerate response

## 2) Type apprentissage à utiliser:



Pour votre projet d'apprentissage utilisant le dataset Spotify Tracks, vous pouvez envisager d'utiliser une approche d'apprentissage supervisé. L'apprentissage supervisé consiste à entraîner un modèle sur un ensemble de données étiquetées (dans ce cas, les chansons avec leur popularité) afin qu'il puisse prédire la popularité des chansons non étiquetées.



Voici quelques algorithmes d'apprentissage supervisé que vous pouvez envisager pour ce projet :

1. **Régression linéaire** : Ce modèle permet de prédire une valeur continue (la popularité) en fonction des caractéristiques musicales. Il est utile lorsque vous souhaitez établir une relation linéaire entre les fonctionnalités et la popularité.
2. **Arbres de décision** : Les arbres de décision peuvent capturer des relations non linéaires entre les fonctionnalités et la popularité. Ils sont faciles à interpréter et peuvent être utilisés pour extraire des règles décisionnelles.
3. **Forêts aléatoires** : Les forêts aléatoires combinent plusieurs arbres de décision pour obtenir des prédictions plus robustes et réduire le surajustement. Elles peuvent être efficaces pour des ensembles de données complexes.
4. **Machines à vecteurs de support (SVM)** : Les SVM sont utilisées pour trouver une séparation optimale entre deux classes. Dans ce cas, vous pouvez les utiliser pour classer les chansons en fonction de leur popularité.
5. **Réseaux de neurones** : Les réseaux de neurones peuvent capturer des relations complexes entre les caractéristiques et la popularité. Ils sont particulièrement adaptés lorsque les données sont très riches et présentent des interactions non linéaires.

### 3) Spécifications du projet:



Certainement ! Voici les spécifications générales pour le projet de prédiction de popularité des chansons :



1. Collecte des données : Téléchargez le jeu de données Spotify Tracks à partir de Kaggle (<https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset>) et familiarisez-vous avec sa structure et ses variables. Assurez-vous de comprendre la signification de chaque attribut.
2. Prétraitement des données : Effectuez une exploration des données et nettoyez-les si nécessaire. Traitez les valeurs manquantes, supprimez les doublons et convertissez les attributs pertinents dans le format approprié. Divisez le jeu de données en ensembles d'entraînement et de test.
3. Sélection des caractéristiques : Identifiez les attributs les plus pertinents pour prédire la popularité des chansons. Vous pouvez utiliser des techniques d'analyse statistique ou d'importance des variables pour effectuer cette sélection. Assurez-vous de conserver les attributs qui sont disponibles au moment de prédire la popularité d'une chanson.
4. Entraînement du modèle : Choisissez un algorithme d'apprentissage automatique adapté à votre problème de prédiction de popularité, comme la régression linéaire, la régression logistique, les arbres de décision, les forêts aléatoires, ou les réseaux de neurones. Entraînez votre modèle en utilisant l'ensemble d'entraînement et les attributs sélectionnés.
5. Évaluation du modèle : Évaluez les performances de votre modèle en utilisant l'ensemble de test. Calculez des métriques telles que l'erreur quadratique moyenne (RMSE), le coefficient de détermination ( $R^2$ ), ou toute autre mesure appropriée pour évaluer la qualité de votre prédiction.
6. Optimisation du modèle : Si les performances de votre modèle ne sont pas satisfaisantes, explorez différentes techniques d'optimisation. Vous pouvez essayer différentes combinaisons d'attributs, utiliser des techniques de réduction de dimension comme l'analyse en composantes principales (PCA), ou ajuster les hyperparamètres de votre algorithme.

### 4) Techniques à utiliser :



Pour votre projet d'apprentissage utilisant le dataset Spotify Tracks, vous pouvez envisager d'utiliser une approche d'apprentissage supervisé. L'apprentissage supervisé consiste à entraîner un modèle sur un ensemble de données étiquetées (dans ce cas, les chansons avec leur popularité) afin qu'il puisse prédire la popularité des chansons non étiquetées.

Voici quelques algorithmes d'apprentissage supervisé que vous pouvez envisager pour ce projet :

1. **Régression linéaire** : Ce modèle permet de prédire une valeur continue (la popularité) en fonction des caractéristiques musicales. Il est utile lorsque vous souhaitez établir une relation linéaire entre les fonctionnalités et la popularité.
2. **Arbres de décision** : Les arbres de décision peuvent capturer des relations non linéaires entre les fonctionnalités et la popularité. Ils sont faciles à interpréter et peuvent être utilisés pour extraire des règles décisionnelles.
3. **Forêts aléatoires** : Les forêts aléatoires combinent plusieurs arbres de décision pour obtenir des prédictions plus robustes et réduire le surajustement. Elles peuvent être efficaces pour des ensembles de données complexes.
4. **Machines à vecteurs de support (SVM)** : Les SVM sont utilisées pour trouver une séparation optimale entre deux classes. Dans ce cas, vous pouvez les utiliser pour classer les chansons en fonction de leur popularité.
5. **Réseaux de neurones** : Les réseaux de neurones peuvent capturer des relations complexes entre les caractéristiques et la popularité. Ils sont particulièrement adaptés lorsque les données sont très riches et présentent des interactions non linéaires.

## 5) Réalisation du projet:

### - Collecte et exploration de données :

1. **Importation des bibliothèques** : Dans votre code, vous devrez importer les bibliothèques nécessaires pour le prétraitement des données, la modélisation et l'évaluation. Des bibliothèques couramment utilisées pour l'apprentissage automatique en Python sont Pandas, NumPy, Scikit-learn et Matplotlib.

python


Copy code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```



2. Chargement des données : Vous devez charger le dataset Spotify Tracks dans votre code. Assurez-vous que le fichier CSV est présent dans votre répertoire de travail.

python

 Copy code


```
data = pd.read_csv("spotify_tracks.csv")
```

pouvez vous m'aider pour l'étape : Exploration des données

Absolument ! Voici quelques étapes pour explorer vos données :

1. Afficher les premières lignes du dataset :

python


 Copy code

```
data.head()
```

Cela affichera les premières lignes du dataset, ce qui vous donnera un aperçu des colonnes et des valeurs.

## 2. Obtenir des informations sur les types de données et les valeurs manquantes :

python


 Copy code

```
data.info()
```

Cela vous fournira des informations sur les types de données de chaque colonne, ainsi que le nombre total de valeurs non nulles. Vous pourrez également identifier les colonnes qui contiennent des valeurs manquantes.

## 3. Obtenir des statistiques descriptives sur les données numériques :

python

 Copy code

```
data.describe()
```

Cette méthode affiche des statistiques descriptives telles que la moyenne, l'écart-type, le minimum, le maximum et les quartiles pour les colonnes numériques de votre dataset.

- Gestion des valeurs manquantes :

Après la transmission des résultats obtenues à chatGPT, il me propose d'effectuer l'une des actions suivantes :

D'après les résultats, il semble que les colonnes "artists", "album\_name" et "track\_name" contiennent chacune une valeur manquante. Le reste des colonnes ne semble pas avoir de valeurs manquantes.

Vous pouvez décider comment gérer ces valeurs manquantes en fonction de la nature de vos données et des objectifs de votre projet. Voici quelques options courantes pour traiter les valeurs manquantes :

1. Supprimer les lignes avec des valeurs manquantes : Si le nombre de lignes avec des valeurs manquantes est relativement faible par rapport à la taille totale de votre dataset, vous pouvez choisir de supprimer ces lignes à l'aide de la méthode `'dropna()'`.
2. Remplir les valeurs manquantes avec une valeur par défaut : Vous pouvez remplacer les valeurs manquantes par une valeur spécifique qui peut être pertinente pour la colonne. Par exemple, pour les colonnes textuelles comme "artists", "album\_name" et "track\_name", vous pouvez utiliser la chaîne de caractères vide ("" ) pour remplacer les valeurs manquantes.
3. Imputer les valeurs manquantes : Si les valeurs manquantes représentent une proportion significative de vos données, vous pouvez utiliser des techniques d'imputation pour estimer ou remplacer les valeurs manquantes. Par exemple, vous pouvez remplacer les valeurs manquantes dans les colonnes numériques par la moyenne ou la médiane des valeurs existantes.

Il est important de prendre en compte le contexte de votre projet et la signification des valeurs manquantes pour prendre une décision appropriée sur la façon de les traiter.


**Remarque** : Il ne prend pas la décision à ma place, mais il me demande de choisir l'option qui est la mieux adaptée au contexte du projet.

Qcq vous proposez ?

Selon les colonnes avec des valeurs manquantes dans votre dataset, voici quelques approches que vous pouvez envisager pour les gérer :

1. Supprimer les lignes avec des valeurs manquantes : Si le nombre de lignes avec des valeurs manquantes est relativement faible par rapport à la taille totale de votre dataset, vous pouvez choisir de supprimer ces lignes à l'aide de la méthode `dropna()`. Par exemple :


python

 Copy code

```
data = data.dropna()
```

2. Remplir les valeurs manquantes avec une valeur par défaut : Si les colonnes avec des valeurs manquantes sont de type texte (comme "artists", "album\_name" et "track\_name"), vous pouvez remplacer les valeurs manquantes par une chaîne de caractères vide ("" ) à l'aide de la méthode `fillna()`. Par exemple :


python

 Copy code

```
data['artists'] = data['artists'].fillna('')
data['album_name'] = data['album_name'].fillna('')
data['track_name'] = data['track_name'].fillna('')
```

3. Imputer les valeurs manquantes : Si vous préférez estimer ou remplacer les valeurs manquantes, vous pouvez utiliser des techniques d'imputation. Par exemple, pour les colonnes numériques, vous pouvez remplacer les valeurs manquantes par la moyenne à l'aide de la méthode `'fillna()'` de Pandas :

python

 Copy code

```
data['column_name'] = data['column_name'].fillna(data['column_name'].mean())
```

Vous devriez remplacer "column\_name" par le nom de la colonne spécifique que vous souhaitez traiter.


Choisissez l'approche qui correspond le mieux à vos besoins et à la nature de vos données. Si vous avez besoin d'une assistance supplémentaire pour implémenter l'approche choisie, n'hésitez pas à me le faire savoir !

#### - Analyse de données :

Afin de mieux comprendre mon jeu de données, ChatGpt m'a proposé d'étudier la relation entre les variables en affichant des graphiques (Histogrammes, matrice de corrélation, ... etc), en affichant la description des différentes features ... etc En transmettant les valeurs obtenues à chatGpt, ou bien simplement les constatations relevées, il est capable de fournir des explications, tirer des conclusions et suggérer des actions à effectuer (Voir le fichier projet\_ml.ipynb).

#### 4. Explorer les distributions des variables numériques :

python


 Copy code

```
data.hist()  
plt.show()
```

Cela générera un histogramme pour chaque variable numérique, vous permettant de visualiser la distribution des données.

#### 5. Explorer les relations entre les variables :

python


 Copy code

```
import seaborn as sns  
sns.pairplot(data)  
plt.show()
```

Cela générera une matrice de diagrammes de dispersion pour toutes les paires de variables numériques, vous permettant de visualiser les relations entre les variables.

#### 6. Explorer les corrélations entre les variables :

python

 Copy code

```
corr_matrix = data.corr()  
sns.heatmap(corr_matrix, annot=True)  
plt.show()
```

Cela générera une carte thermique (heatmap) des corrélations entre les variables, vous permettant de visualiser les relations linéaires entre les variables.

- **Exemple** : Matrice de corrélation

comment puis lire la heatmap obtenu

Pour lire la heatmap obtenue, vous devez interpréter les couleurs et les valeurs affichées dans la heatmap. Voici comment vous pouvez le faire :


1. Couleurs : La heatmap utilise une échelle de couleurs pour représenter les valeurs. Par défaut, les valeurs plus élevées sont représentées par des couleurs plus foncées et les valeurs plus faibles par des couleurs plus claires. Vous pouvez vous référer à la légende de couleur sur le côté de la heatmap pour comprendre quelles valeurs correspondent à quelles couleurs.
2. Valeurs annotées : Les valeurs numériques correspondantes à chaque paire de variables sont annotées à l'intérieur de chaque cellule de la heatmap. Ces valeurs indiquent le coefficient de corrélation entre les deux variables. Une valeur proche de 1 indique une corrélation positive forte, une valeur proche de -1 indique une corrélation négative forte, et une valeur proche de 0 indique une corrélation faible ou inexistante.

En analysant la matrice de corrélation, voici quelques observations et conclusions que l'on peut tirer :

- La colonne "popularity" est corrélée positivement avec les colonnes "loudness", "energy" et "danceability". Cela suggère que les chansons avec des niveaux sonores plus élevés, plus d'énergie et plus de capacité de danse ont tendance à être plus populaires.
- La colonne "popularity" est également corrélée négativement avec la colonne "acousticness". Cela indique que les chansons moins acoustiques ont tendance à être plus populaires.
- D'autres colonnes telles que "explicit", "speechiness" et "instrumentalness" montrent des corrélations faibles avec la popularité.
- Il y a également des corrélations entre certaines colonnes indépendantes, par exemple, une corrélation positive entre "loudness" et "energy".

Ces observations peuvent être utiles dans la sélection des colonnes pertinentes pour la prédiction de la popularité. Vous pouvez envisager de choisir les colonnes "loudness", "energy", "danceability" et "acousticness" comme variables indépendantes pour votre modèle de prédiction. Cependant, il est important de noter que la corrélation ne signifie pas forcément une relation de cause à effet, et d'autres facteurs peuvent également influencer la popularité des chansons.

N'oubliez pas que cette interprétation est basée uniquement sur la corrélation entre les variables et ne garantit pas la précision ou la performance du modèle de prédiction. Il est recommandé de réaliser des analyses plus approfondies et d'utiliser des techniques d'apprentissage automatique appropriées pour construire votre modèle de prédiction de popularité.

 Regenerate response

## - Traitement des données:



D'après la description des variables, les colonnes nécessitant un encodage sont :

- artists (les noms des artistes)
- album\_name (les noms des albums)
- track\_name (les noms des pistes)
- explicit (variable binaire indiquant si les paroles sont explicites)
- track\_genre (le genre musical de la piste)

Pour traiter les variables catégorielles, vous pouvez utiliser l'encodage one-hot (dummy encoding) pour les transformer en variables binaires. Voici le code pour effectuer cette opération en utilisant la fonction `get_dummies` de pandas :

```
python Copy code

# Sélection des variables catégorielles dans le DataFrame 'data'
variables_catégorielles = ['track_id', 'artists', 'album_name', 'track_name']

# Encodage one-hot des variables catégorielles
data_encoded = pd.get_dummies(data, columns=variables_catégorielles)

# Affichage du DataFrame encodé
print(data_encoded.head())
```

aidez moi pour le : Traitement des variables booléennes

Pour le traitement des variables booléennes dans votre DataFrame, vous pouvez utiliser l'encodage binaire. Vous pouvez mapper la valeur booléenne `True` à 1 et la valeur `False` à 0.

Voici un exemple de code pour effectuer cette opération :

```
python Copy code

# Traitement des variables booléennes
data['explicit'] = data['explicit'].astype(int)
```


## - Division des données :

En résumé, une fois que vos données ont été nettoyées et que l'encodage des données catégorielles/booléennes a été effectué, voici les étapes suivantes pour votre projet de prédiction de popularité des chansons :

1. Séparez vos données en ensembles d'entraînement, de validation et de test.
2. Normalisez les caractéristiques si nécessaire.
3. Choisissez un algorithme de prédiction adapté à votre problème.
4. Entraînez votre modèle en utilisant l'ensemble d'entraînement.
5. Réglez les hyperparamètres de l'algorithme en utilisant l'ensemble de validation.
6. Évaluez les performances de votre modèle en utilisant l'ensemble de test.
7. Effectuez des ajustements supplémentaires si nécessaire.
8. Interprétez les résultats obtenus.

Voici un exemple de code en Python pour diviser vos données en ensembles d'entraînement et de test à l'aide de la bibliothèque scikit-learn :

python

 Copy code

```
from sklearn.model_selection import train_test_split

# Supposons que vous avez vos données dans les variables X (caractéristiques) et y (cibles)

# Divisez les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Explications :


## - Entraînement des modèles :

Maintenant que vos données sont divisées en ensembles d'entraînement et de test, la prochaine étape consiste à entraîner votre modèle sur l'ensemble d'entraînement et à évaluer sa performance sur l'ensemble de test. Voici les étapes à suivre :

1. **Entraînement du modèle** : Utilisez l'ensemble d'entraînement pour entraîner votre modèle en ajustant les poids et les paramètres du modèle. Cela se fait généralement à l'aide d'un algorithme d'optimisation, tel que la descente de gradient, qui minimise une fonction de perte ou d'erreur.
2. **Validation du modèle** : Après chaque itération d'entraînement, évaluez les performances de votre modèle sur l'ensemble de validation. Cela vous permet d'ajuster les hyperparamètres du modèle, tels que le taux d'apprentissage, le nombre d'itérations, la taille du lot (batch size), etc., pour améliorer les performances du modèle.
3. **Évaluation du modèle** : Une fois que vous avez terminé l'entraînement et ajusté les hyperparamètres, évaluez la performance finale de votre modèle sur l'ensemble de test. Cela vous donnera une estimation réaliste de la capacité de généralisation du modèle et de sa performance sur de nouvelles données.

Voici un exemple pour vous montrer comment utiliser ces étapes :

python

 Copy code

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Choix du modèle (par exemple, régression linéaire)
model = LinearRegression()

# Entraînement du modèle
model.fit(X_train_pca, y_train)

# Prédiction sur les données de test
y_pred = model.predict(X_test_pca)

# Évaluation du modèle (par exemple, calcul de l'erreur quadratique moyenne)
mse = mean_squared_error(y_test, y_pred)
print("Erreur quadratique moyenne :", mse)
```


Dans cet exemple, nous utilisons une régression linéaire comme modèle. Après l'entraînement du modèle et la prédiction sur les données de test, nous évaluons les performances du modèle en calculant l'erreur quadratique moyenne entre les valeurs prédites (`y\_pred`) et les valeurs réelles (`y\_test`).

donnez moi d'autres modèles à utiliser avec le code associé

Voici quelques modèles de régression que vous pouvez essayer avec le code associé :

### 1. Régression Ridge :

python

 Copy code

```
from sklearn.linear_model import Ridge


# Créer un modèle de régression Ridge
model = Ridge(alpha=0.5)

# Entraîner le modèle sur les données d'apprentissage réduites en dimensions
model.fit(X_train_pca, y_train)

# Prédire les valeurs de popularité sur les données de test réduites en dimensions
y_pred = model.predict(X_test_pca)
```

### 2. Régression Lasso :

python

 Copy code

```
from sklearn.linear_model import Lasso
```

a part les modèles de régression, quels autres modèles proposez vous ?

En plus des modèles de régression, voici quelques autres modèles populaires que vous pouvez essayer pour votre tâche de prédiction de la popularité :


1. Modèles d'arbres de décision : Les arbres de décision et leurs variantes comme les forêts aléatoires (Random Forests) et les arbres de décision boostés (Gradient Boosting) peuvent être efficaces pour la prédiction de variables continues comme la popularité.
2. Modèles de réseaux neuronaux : Les réseaux neuronaux, en particulier les réseaux de neurones profonds (Deep Learning), sont utilisés dans de nombreux domaines et peuvent être appliqués à des tâches de prédiction de variables continues.
3. Méthodes de support vector regression (SVR) : Les méthodes de support vector regression sont basées sur les machines à vecteurs de support et peuvent être utilisées pour modéliser des problèmes de régression.
4. Méthodes d'apprentissage par ensemble : Les méthodes d'apprentissage par ensemble, telles que le bagging, le boosting et le stacking, combinent plusieurs modèles pour améliorer les performances prédictives.
5. Modèles de régression non linéaire : Si votre problème présente des relations non linéaires, vous pouvez essayer des modèles de régression non linéaire, tels que les machines à vecteurs de support à noyau (SVM kernel), les processus gaussiens (Gaussian Processes), ou les réseaux neuronaux avec des fonctions d'activation non linéaires.

donnez moi le code correspondant à chaque modele

Voici le code correspondant à chaque modèle :

1. Modèle d'arbre de décision (Decision Tree) :

python

 Copy code

```
from sklearn.tree import DecisionTreeRegressor


# Créer un modèle d'arbre de décision
model = DecisionTreeRegressor()

# Entraîner le modèle sur les données d'apprentissage réduites en dimensions
model.fit(X_train_pca, y_train)

# Prédire les valeurs de popularité sur les données de test réduites en dimensions
y_pred = model.predict(X_test_pca)


mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Erreur quadratique moyenne :", mse)
print("Erreur absolue moyenne :", mae)
print("Coefficient de détermination (R-squared) :", r2)
```

 Regenerate response

## 2. Modèle de forêt aléatoire (Random Forest) :

python

 Copy code

```
from sklearn.ensemble import RandomForestRegressor

# Créer un modèle de forêt aléatoire
model = RandomForestRegressor()

# Entraîner le modèle sur les données d'apprentissage réduites en dimensions
model.fit(X_train_pca, y_train)

# Prédire les valeurs de popularité sur les données de test réduites en dimensions
y_pred = model.predict(X_test_pca)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Erreur quadratique moyenne :", mse)
print("Erreur absolue moyenne :", mae)
print("Coefficient de détermination (R-squared) :", r2)
```

### - Métriques d'évaluation :



quelles autres métrique puis je utiliser ?

Voici quelques métriques couramment utilisées pour évaluer les performances des modèles de régression :

1. Erreur quadratique moyenne (Mean Squared Error, MSE) :

- Calcul de l'erreur quadratique moyenne entre les valeurs prédites et les valeurs réelles.
- Plus la valeur est faible, meilleure est la performance du modèle.
- Disponible dans scikit-learn : `from sklearn.metrics import mean_squared_error`

2. Erreur absolue moyenne (Mean Absolute Error, MAE) :

- Calcul de l'erreur absolue moyenne entre les valeurs prédites et les valeurs réelles.
- Plus la valeur est faible, meilleure est la performance du modèle.
- Disponible dans scikit-learn : `from sklearn.metrics import mean_absolute_error`


3. Coefficient de détermination (R-squared) :

- Mesure la proportion de la variance totale de la variable cible expliquée par le modèle.
- Une valeur proche de 1 indique une bonne adéquation du modèle.
- Disponible dans scikit-learn : `from sklearn.metrics import r2_score`

- **Choix du modèle:**

Voici le code pour créer un graphique permettant de distinguer le meilleur modèle :

python

 Copy code

```
import matplotlib.pyplot as plt

models = ['Régression linéaire', 'Régression Lasso', 'ElasticNet', 'Arbre de
mse = [484.55687929316156, 493.6092501190866, 492.8254847302422, 434.0932456

# Tracer le graphique
plt.bar(models, mse)
plt.xlabel('Modèles')
plt.ylabel('Erreur quadratique moyenne')
plt.title('Comparaison de l\'erreur quadratique moyenne des modèles')
plt.xticks(rotation=45)

# Afficher les valeurs au-dessus de chaque barre
for i in range(len(models)):
    plt.text(i, mse[i], str(mse[i]), ha='center', va='bottom')

# Afficher le graphique
plt.show()
```

## - Conclusions ChatGPT:

Merci d'avoir fourni les résultats des différents modèles. Voici quelques suggestions pour améliorer l'efficacité :

1. Régression linéaire : La performance de la régression linéaire est assez faible avec un faible coefficient de détermination (R-squared). Vous pouvez essayer d'explorer d'autres modèles plus complexes ou utiliser des techniques de feature engineering pour améliorer la performance.
2. Régression Lasso et ElasticNet : Ces modèles semblent avoir une performance similaire à la régression linéaire. Vous pouvez également explorer d'autres modèles plus avancés.
3. Decision Tree : Le modèle de l'arbre de décision a une performance légèrement meilleure que la régression linéaire, mais il est encore possible d'améliorer les résultats. Vous pouvez essayer d'ajuster les hyperparamètres de l'arbre de décision, comme la profondeur maximale de l'arbre ou le nombre minimum d'échantillons requis pour diviser un nœud.
4. Random Forest : Le modèle de Random Forest montre une performance relativement meilleure que les modèles précédents. Vous pouvez essayer d'optimiser les hyperparamètres du Random Forest, tels que le nombre d'estimateurs et la profondeur maximale de chaque arbre, pour voir s'il y a une amélioration supplémentaire possible.


- **Amélioration des résultats obtenues:**

Pour améliorer les résultats de vos modèles de prédiction de popularité des chansons, voici quelques suggestions :

1. Explorer d'autres modèles (GradientBoosting) : En plus des modèles que vous avez déjà essayés, comme la régression linéaire ou les réseaux de neurones, vous pouvez explorer des modèles tels que le GradientBoosting. Le GradientBoosting est une technique d'apprentissage automatique ensembliste qui combine plusieurs modèles plus faibles pour améliorer les performances de prédiction.
2. Réduction de dimensionnalité (PCA) : Si vous avez un grand nombre de caractéristiques dans votre jeu de données, vous pouvez utiliser des techniques de réduction de dimensionnalité comme l'Analyse en Composantes Principales (PCA). Cela vous permet de réduire la dimension de vos données en conservant les informations les plus importantes. En réduisant le nombre de caractéristiques, vous pouvez simplifier votre modèle et potentiellement améliorer ses performances.
3. Ajustement des hyperparamètres du modèle : Les hyperparamètres sont des paramètres externes au modèle qui contrôlent son comportement. Ils peuvent inclure des paramètres tels que le taux d'apprentissage, la profondeur de l'arbre de décision, le nombre d'itérations, etc. L'ajustement des hyperparamètres peut jouer un rôle crucial dans l'optimisation des performances de votre modèle. Vous pouvez utiliser des techniques telles que la recherche par grille (grid search) ou l'optimisation bayésienne pour trouver les meilleurs hyperparamètres pour votre modèle.

Vous pouvez utiliser la classe `PCA` de scikit-learn pour effectuer une analyse en composantes principales (PCA) et réduire la dimensionnalité des caractéristiques. Voici un exemple :

python

 Copy code


```
from sklearn.decomposition import PCA

# Instancier l'objet PCA et spécifier le nombre de composantes souhaité
pca = PCA(n_components=2)

# Réduire la dimensionnalité des caractéristiques
X_pca = pca.fit_transform(X)
```

Voici un exemple de code pour effectuer une recherche d'hyperparamètres pour le modèle Random Forest en utilisant la méthode de recherche par grille (grid search) avec validation croisée :

python

 Copy code

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Définir les hyperparamètres à optimiser
param_grid = {
    'n_estimators': [100, 200, 300], # Nombre d'arbres dans la forêt
    'max_depth': [None, 5, 10], # Profondeur maximale de chaque arbre
    'min_samples_split': [2, 5, 10], # Nombre minimum d'échantillons requis
    'min_samples_leaf': [1, 2, 4] # Nombre minimum d'échantillons requis po
}

# Créer un modèle Random Forest
model = RandomForestRegressor()

# Effectuer la recherche d'hyperparamètres avec validation croisée
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, sco
grid_search.fit(X_train, y_train)

# Afficher les meilleurs hyperparamètres et la meilleure performance
print("Meilleurs hyperparamètres :", grid_search.best_params_)
print("Meilleur coefficient de détermination (R-squared) :", grid_search.bes

# Utiliser le modèle avec les meilleurs hyperparamètres pour faire des prédi
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

Le code p précédent s'exécute sans donner de résultat.

- Solutions proposées par ChatGPT:

Si votre dataset est trop volumineux et que le code s'exécute sans fournir de résultats, vous pouvez envisager les options suivantes pour réduire la taille du dataset et effectuer l'optimisation des hyperparamètres :




1. Échantillonnage aléatoire : Au lieu d'utiliser l'ensemble complet de données, vous pouvez effectuer un échantillonnage aléatoire pour sélectionner un sous-ensemble plus petit à utiliser pour l'optimisation des hyperparamètres. Par exemple, vous pouvez utiliser la fonction `sample` de pandas pour obtenir un échantillon aléatoire de lignes de votre dataframe.
2. Échantillonnage stratifié : Si votre dataset est déséquilibré en termes de répartition des classes ou des catégories que vous souhaitez prédire, vous pouvez effectuer un échantillonnage stratifié pour préserver la proportion des différentes classes ou catégories dans votre sous-ensemble.
3. Division en batches : Si votre dataset est très volumineux, vous pouvez diviser votre ensemble de données en batches plus petits et effectuer l'optimisation des hyperparamètres sur chaque batch individuellement. Cela peut être fait en utilisant la fonction `array_split` de numpy pour diviser votre ensemble de données en plusieurs parties.
4. Utilisation d'une machine plus puissante : Si vous disposez d'une machine avec plus de ressources, vous pouvez envisager d'augmenter la capacité de calcul en utilisant une machine avec plus de mémoire ou en utilisant des techniques de parallélisme pour accélérer le processus.

Il est important de noter que la réduction de la taille du dataset peut entraîner une perte d'information et une diminution de la précision du modèle. Par conséquent, il est recommandé de trouver un équilibre entre la taille du dataset et les ressources de calcul disponibles pour obtenir des résultats satisfaisants.

## 1. Échantillonnage aléatoire :

python

 Copy code

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Définir la taille de l'échantillon
sample_size = 1000

# Effectuer un échantillonnage aléatoire de votre dataframe
sampled_data = X_train.sample(n=sample_size, random_state=42)
sampled_targets = y_train.loc[sampled_data.index]

# Définir les hyperparamètres à optimiser
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}


# Instancier le modèle RandomForest
rf = RandomForestRegressor()

# Effectuer une recherche sur grille pour trouver les meilleures valeurs d'h
grid_search = GridSearchCV(rf, param_grid=param_grid, cv=5)
grid_search.fit(sampled_data, sampled_targets)

# Obtenir les meilleurs hyperparamètres
best_params = grid_search.best_params_
print("Meilleurs paramètres :", best_params)
```

## 2. Division en batches :

python

 Copy code

```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Définir la taille de chaque batch
batch_size = 1000

# Diviser votre ensemble de données en batches
batches = np.array_split(X_train, len(X_train) // batch_size)
target_batches = np.array_split(y_train, len(X_train) // batch_size)


# Définir les hyperparamètres à optimiser
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

# Instancier le modèle RandomForest
rf = RandomForestRegressor()

# Effectuer une recherche sur grille pour trouver les meilleures valeurs d'h
best_params = None
best_score = float('-inf')

for i, batch in enumerate(batches):
    target_batch = target_batches[i]

    # Instancier le modèle
    rf = RandomForestRegressor()
```

 Regenerate response