République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tunis El Manar
École Nationale d'Ingénieurs de Tunis

Département Génie Industriel

Projet de Data Mining

Analyse des Tendances Musicales et Recommandations Personnalisées.

Elaboré par :

Yassine Soussi

Hiba Bchatnia

2ème Année GI

Année universitaire : 2024/2025

Table des matières

Table des figures

Introduction

Ce rapport se concentre sur l'analyse des tendances musicales et la formulation de recommandations personnalisées en exploitant un ensemble de données détaillé couvrant plus de 600 000 chansons disponibles sur Spotify. Ce dataset comprend des caractéristiques audio variées ainsi que des métadonnées pertinentes, offrant une base solide pour l'extraction d'informations significatives. Le projet englobe plusieurs étapes clés, notamment la collecte et la préparation des données, l'analyse exploratoire, la modélisation prédictive à l'aide de deux algorithmes : les Forêts aléatoires (Random Forest) et la Régression linéaire, ainsi que la validation des modèles, l'interprétation des résultats et la formulation de recommandations adaptées.

En outre, ce projet s'inscrit dans un contexte plus large où l'importance de la maintenance prédictive est cruciale. Bien que souvent associée aux domaines industriels, la maintenance prédictive joue également un rôle clé dans les systèmes de recommandation. Elle permet d'anticiper les évolutions des préférences des utilisateurs en identifiant des modèles dans les données. Cela optimise non seulement l'expérience utilisateur, mais favorise également la pérennité et la performance des systèmes automatisés en réduisant les inefficacités et en augmentant la satisfaction globale des utilisateurs.

Les applications concrètes de la maintenance prédictive sont variées et étendues. Dans le domaine industriel, elle est utilisée pour surveiller l'état des machines et éviter les pannes coûteuses en anticipant les besoins de maintenance. Dans les transports, elle permet d'assurer la sécurité des véhicules en identifiant les problèmes mécaniques avant qu'ils ne surviennent. En santé, elle facilite le suivi des patients grâce à des outils prédictifs pour détecter les risques de maladies ou complications. Dans le contexte des systèmes de recommandation, comme ceux explorés dans ce projet, la maintenance prédictive garantit une mise à jour continue des modèles, la détection des biais, et l'adaptation des algorithmes aux évolutions des comportements utilisateurs. Ces applications démontrent l'importance stratégique de cette approche dans divers secteurs pour garantir performance et fiabilité.

Chapitre 1

Méthodologie

1.1 Préparation des données et sélection des caractéristiques

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
```

FIGURE 1.1 – Extrait de code : Préparation des données et sélection des caractéristiques

Dans cette section du code, plusieurs bibliothèques sont importées pour préparer, analyser et modéliser les données issues du dataset Spotify. Voici les étapes clés couvertes par ces imports :

— **Imputation des données manquantes** : Le module `SimpleImputer` de `sklearn.impute` est utilisé pour traiter les valeurs manquantes dans le dataset. Cela garantit que les données sont complètes et prêtes pour l'entraînement des modèles. Par exemple, les valeurs nulles ou manquantes peuvent être remplacées par la moyenne, la médiane ou d'autres stratégies définies par l'utilisateur.

— **Modélisation avec les Forêts aléatoires** : L'algorithme `RandomForestRegressor` de `sklearn.ensemble` est importé pour effectuer des tâches de régression. Cet algorithme, basé sur un ensemble d'arbres de décision, est particulièrement adapté à des données avec des relations complexes et non linéaires.

— **Analyse exploratoire et visualisation** :

  — `pandas` et `numpy` : Ces bibliothèques sont utilisées pour manipuler et analyser les données. `pandas` permet de gérer les données sous forme de tableaux, tandis que `numpy` fournit des outils pour les opérations numériques rapides.

  — `matplotlib.pyplot` et `seaborn` : Ces bibliothèques facilitent la visualisation des données grâce à des graphiques, tels que des histogrammes ou des heatmaps, pour identifier les tendances ou les corrélations dans les données.

— **Division des données** : La fonction `train_test_split` de `sklearn.model_selection` est utilisée pour diviser le dataset en ensembles d'entraînement et de test. Cela permet d'entraîner les modèles sur une partie des données et de les évaluer sur une autre partie pour garantir leur capacité à généraliser.

— **Normalisation et sélection des caractéristiques** :

  — `StandardScaler` : Ce module normalise les données en ajustant leur moyenne et leur écart type, ce qui est particulièrement important pour les modèles sensibles à l'échelle des données, comme les Machines à vecteurs de support.

  — `SelectKBest` et `f_regression` : Ces outils permettent de sélectionner les K meilleures caractéristiques (*features*) en fonction de leur corrélation avec la variable cible, en utilisant des tests statistiques comme la régression F. Cela réduit la complexité du modèle et améliore ses performances.

Afficher les colonnes du dataset pour identifier la colonne cible, ce bloc de code prépare ainsi les bases pour une analyse approfondie des données, la sélection des caractéristiques les plus pertinentes et l'entraînement de modèles robustes.

1.2 Demande de la colonne cible pour la modélisation

```
# Demander à l'utilisateur de saisir le nom exact de la colonne cible
target_column = input("\nEntrez le nom exact de la colonne cible (par exemple 'popularity') : ")
# Vérifier si la colonne cible existe
if target_column not in dataset.columns:
    raise ValueError(f"La colonne cible '{target_column}' n'existe pas dans le dataset. Veuillez vérifier.")
```

FIGURE 1.2 – Extrait de code : Demande de la colonne cible pour la modélisation

Apprés afficher les colonnes du dataset pour identifier la colonne cible, l'utilisateur est invité à spécifier le nom de la colonne cible sur laquelle il souhaite effectuer des prédictions. Cette colonne représente la variable à prédire par les modèles d'apprentissage automatique. Voici le fonctionnement détaillé :

— **Demande du nom de la colonne cible** : Le code commence par demander à l'utilisateur de saisir le nom exact de la colonne cible à l'aide de la fonction `input()`. Par exemple, si l'utilisateur souhaite prédire la popularité des chansons, il peut entrer `'popularity'` comme nom de colonne.

— **Vérification de l'existence de la colonne** : Après que l'utilisateur ait saisi le nom de la colonne, le code vérifie si cette colonne existe réellement dans le dataset. Si la colonne n'est pas présente, une erreur est levée via `ValueError`, informant l'utilisateur que la colonne cible est invalide et lui demandant de vérifier le nom de la colonne. Cela permet de s'assurer que la modélisation se fait sur une colonne correcte.

Cette étape est cruciale pour éviter les erreurs qui pourraient survenir si l'utilisateur entre un nom incorrect ou inexistant pour la colonne cible.

1.3 Aperçu des données

```
# Étape 3 : Aperçu des données
print("\nAperçu des premières lignes du dataset :")
print(dataset.head())
print("\nInformations sur le dataset :")
print(dataset.info())
```

FIGURE 1.3 – Extrait de code : Aperçu des données

Dans cette étape, nous effectuons un premier aperçu du dataset afin de mieux comprendre sa structure, les types de données qu'il contient, et d'obtenir des informations générales sur les enregistrements. Voici les étapes détaillées :

— **Affichage des premières lignes du dataset** : La fonction `head()` est utilisée pour afficher les premières lignes du dataset. Cela permet de vérifier rapidement les données et de s'assurer qu'elles ont été chargées correctement. Cela permet également d'observer les premières valeurs de chaque colonne pour mieux comprendre les informations contenues dans le dataset.

— **Informations générales sur le dataset** : La fonction `info()` fournit des informations détaillées sur le dataset, telles que le nombre total d'entrées, le nombre de valeurs non-nulles par colonne, et les types de données associés à chaque colonne. Ces informations sont essentielles pour détecter les valeurs manquantes et pour mieux comprendre la structure des données.

Cette étape est cruciale pour la préparation des données et pour identifier d'éventuelles anomalies ou lacunes dans le dataset avant d'entamer les étapes de modélisation.

1.4 Vérification des valeurs manquantes

```
# Étape 4 : Vérification des valeurs manquantes
missing_values = dataset.isnull().sum()
print("\nValeurs manquantes par colonne :")
print(missing_values)
```

FIGURE 1.4 – Extrait de code : Vérification des valeurs manquantes

L'une des étapes cruciales dans la préparation des données consiste à vérifier la présence de valeurs manquantes. Dans cette section, nous utilisons une méthode pour compter et afficher le nombre de valeurs manquantes par colonne. Voici les détails du code :

— **Vérification des valeurs manquantes** : La fonction `isnull()` est utilisée pour identifier les valeurs manquantes dans le dataset. Cette fonction renvoie un DataFrame où chaque cellule est marquée comme `True` si la valeur est manquante, et `False` sinon.

— **Somme des valeurs manquantes par colonne** : En appliquant la fonction `sum()` sur le résultat de `isnull()`, nous obtenons le nombre total de valeurs manquantes pour chaque colonne. Cela permet de savoir rapidement quelles colonnes contiennent des données manquantes et de décider de la manière de les traiter (par exemple, en imputant les valeurs manquantes ou en supprimant les colonnes concernées).

Cette étape permet de détecter et d'identifier les valeurs manquantes dans le dataset, ce qui est essentiel pour garantir la qualité des données avant l'entraînement des modèles de machine learning.

1.5 Traitement des colonnes non numériques et valeurs manquantes

```python
# Étape 5 : Traitement des colonnes non numériques et valeurs manquantes
# Identifier les colonnes non numériques
non_numeric_columns = dataset.select_dtypes(include=['object']).columns
if len(non_numeric_columns) > 0:
    print("\nColonnes non numériques détectées :", list(non_numeric_columns))
    # Remplir les colonnes non numériques avec une valeur par défaut (exemple : "Inconnu")
    dataset[non_numeric_columns] = dataset[non_numeric_columns].fillna("Inconnu")
```

FIGURE 1.5 – Extrait de code : Traitement des colonnes non numériques et valeurs manquantes

Dans cette étape, nous traitons les colonnes non numériques ainsi que les valeurs manquantes dans le dataset. Cela est particulièrement important pour assurer que toutes les colonnes sont prêtes pour le modèle d'apprentissage automatique, qui nécessite souvent des données numériques et complètes. Voici les actions effectuées :

— **Identification des colonnes non numériques** : La fonction `select_dtypes(include=['object'])` est utilisée pour identifier toutes les colonnes qui contiennent des données non numériques (par exemple, des chaînes de caractères). Ces colonnes peuvent inclure des informations textuelles, telles que les noms des artistes ou des genres musicaux.

— **Remplissage des valeurs manquantes dans les colonnes non numériques** : Si des colonnes non numériques sont détectées, le code remplit les valeurs manquantes par une valeur par défaut, ici `"Inconnu"`. Cela permet d'éviter les erreurs lors de l'entraînement des modèles, car les valeurs manquantes dans les données textuelles pourraient entraîner des problèmes.

Ce processus permet de garantir que toutes les colonnes sont cohérentes et prêtes à être utilisées pour les étapes suivantes de l'analyse et de la modélisation.

1.6 Traitement des valeurs manquantes dans les colonnes numériques

```
# Remplir les colonnes numériques avec leur moyenne
numeric_columns = dataset.select_dtypes(include=['float64', 'int64']).columns
if len(numeric_columns) > 0:
    dataset[numeric_columns] = dataset[numeric_columns].fillna(dataset[numeric_columns].mean())

print("\nValeurs manquantes traitées.")
```

FIGURE 1.6 – Extrait de code :Traitement des valeurs manquantes dans les colonnes numériques

Dans cette étape, nous traitons les valeurs manquantes spécifiquement dans les colonnes numériques du dataset. Il est important de traiter les valeurs manquantes dans les données numériques de manière appropriée afin d'éviter des erreurs lors de l'entraînement des modèles. Voici les actions entreprises :

- **Identification des colonnes numériques** : La fonction `select_dtypes(include=['float64', 'int64'])` est utilisée pour identifier toutes les colonnes contenant des valeurs numériques (entiers et flottants). Ces colonnes peuvent inclure des caractéristiques telles que la popularité, la durée, ou des métriques audio comme l'énergie et la danseabilité.
- **Remplissage des valeurs manquantes par la moyenne** : Pour chaque colonne numérique contenant des valeurs manquantes, celles-ci sont remplacées par la moyenne de cette colonne. Cela est fait en utilisant la fonction `fillna()` et en appliquant la moyenne calculée avec `mean()`. Cette méthode permet de maintenir l'intégrité des données tout en remplissant les valeurs manquantes de manière raisonnable.

Ce traitement assure que les colonnes numériques n'ont plus de valeurs manquantes, ce qui est essentiel pour l'entraînement des modèles de machine learning.

1.7 Statistiques Descriptives

```
# **Étape 6 : Statistiques Descriptives**
print("\nStatistiques descriptives :")
print(dataset[numeric_columns].describe())
```

FIGURE 1.7 – Extrait de code : Statistiques Descriptives

Dans cette étape, nous générons un résumé statistique des colonnes numériques du dataset. Cela permet d'obtenir une vue d'ensemble des principales caractéristiques numériques, telles que leur distribution, leur dispersion, et leurs valeurs extrêmes. Voici les détails de l'analyse :

- **Affichage des statistiques descriptives** : La fonction `describe()` est appliquée aux colonnes numériques du dataset. Cette fonction génère un tableau résumant plusieurs mesures clés, telles que :
  - **Nombre d'entrées non nulles** (`count`) : Indique le nombre de valeurs présentes dans chaque colonne.
  - **Moyenne** (`mean`) : Représente la valeur moyenne de chaque colonne.
  - **Écart-type** (`std`) : Mesure la dispersion des données autour de la moyenne.
  - **Valeurs minimales et maximales** (`min` et `max`) : Indiquent les bornes des valeurs dans chaque colonne.
  - **Quartiles** (`25%, 50%, 75%`) : Fournissent une mesure de la répartition des données.
- **Utilité de cette étape** : Ces statistiques descriptives permettent d'identifier les tendances générales et les éventuelles anomalies ou valeurs aberrantes dans le dataset, facilitant ainsi une meilleure compréhension des données avant d'entamer la modélisation.

Cette étape est essentielle pour explorer les données et s'assurer que leur distribution est cohérente avec les attentes pour la suite des analyses.

1.8 Calcul des Moyennes, Médianes et Écarts-types

```python
# Calculer les moyennes, médianes et écarts-types pour chaque variable
mean = dataset[numeric_columns].mean()
median = dataset[numeric_columns].median()
std_dev = dataset[numeric_columns].std()
print("\nMoyennes :")
print(mean)

print("\nMédianes :")
print(median)

print("\nÉcarts-types :")
print(std_dev)
```

FIGURE 1.8 – Extrait de code : Calcul des Moyennes, Médianes et Écarts-types

Dans cette étape, nous calculons trois mesures statistiques fondamentales pour chaque variable numérique du dataset : la moyenne, la médiane et l'écart-type. Ces mesures permettent de mieux comprendre la répartition des données et leur variabilité. Voici les détails de chaque mesure :

- **Moyennes** : La moyenne, calculée à l'aide de la fonction `mean()`, représente la valeur moyenne de chaque variable. Elle donne une idée générale des tendances centrales des données.
- **Médianes** : La médiane, calculée à l'aide de la fonction `median()`, est la valeur centrale des données lorsqu'elles sont triées. Elle est moins sensible aux valeurs aberrantes que la moyenne et fournit un indicateur robuste de la tendance centrale.
- **Écarts-types** : L'écart-type, calculé avec la fonction `std()`, mesure la dispersion des données par rapport à la moyenne. Une faible valeur d'écart-type indique que les données sont concentrées autour de la moyenne, tandis qu'une valeur élevée reflète une plus grande variabilité.

Utilité de cette analyse : Ces mesures statistiques permettent d'identifier les caractéristiques principales des données et de détecter les éventuelles anomalies ou irrégularités, ce qui est crucial pour la préparation des données avant la modélisation.

Les résultats des calculs sont affichés pour chaque variable numérique, fournissant une vue complète des tendances et de la variabilité des données.

1.9 Ingénierie des Caractéristiques : Ajout de Moyennes Mobiles

```
# Étape 6 : Ingénierie des caractéristiques
# Exemple d'ajout de moyennes mobiles
window_size = 3  # Taille de la fenêtre pour la moyenne mobile
dataset['moving_avg'] = dataset[target_column].rolling(window=window_size).mean()
```

FIGURE 1.9 – Extrait de code : Ingénierie des Caractéristiques : Ajout de Moyennes Mobiles

L'ingénierie des caractéristiques est une étape clé dans la préparation des données, permettant d'extraire des informations supplémentaires à partir des variables existantes. Dans cette étape, nous illustrons l'ajout d'une nouvelle caractéristique au dataset, basée sur le calcul de moyennes mobiles.

— **Définition des moyennes mobiles** : Une moyenne mobile est une méthode de lissage qui calcule la moyenne d'un ensemble de valeurs sur une fenêtre glissante de taille fixée. Dans ce cas, une fenêtre de taille 3 (`window_size = 3`) est utilisée pour calculer la moyenne des trois dernières observations pour chaque valeur de la colonne cible (`target_column`).

— **Mise en œuvre** : La méthode `rolling(window=window_size).mean()` est appliquée à la colonne cible pour générer une nouvelle colonne, appelée `moving_avg`. Cette colonne contient les moyennes mobiles pour chaque observation, permettant de capturer les tendances locales dans les données.

— **Avantages des moyennes mobiles** :

  — Elles aident à atténuer les fluctuations aléatoires dans les données.

  — Elles mettent en évidence les tendances à court terme qui peuvent être importantes pour des tâches prédictives.

  — Elles enrichissent les données avec une caractéristique supplémentaire qui peut améliorer la performance des modèles de machine learning.

L'intégration des moyennes mobiles constitue un exemple d'ingénierie des caractéristiques, visant à améliorer la richesse et la pertinence des données utilisées pour l'apprentissage.

1.10 Préparation des Données : Normalisation et Division des Ensembles

```python
# Supprimer la colonne 'Unnamed: 0' si elle est présente
dataset = dataset.drop(columns=['Unnamed: 0'], errors='ignore')
```

```python
# Séparer les features (X) et la target (y)
X = dataset.drop(columns=[target_column])
y = dataset[target_column]
```

```python
# Sélectionner uniquement les colonnes numériques pour X
X_numeric = X.select_dtypes(include=['float64', 'int64'])
```

```python
# Étape 7 : Normalisation des données
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_numeric)  # Normaliser uniquement les colonnes numériques
```

```python
# Étape 8 : Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```python
# Étape 9 : Résumé des ensembles
print("\nTaille de l'ensemble d'entraînement :", X_train.shape)
print("Taille de l'ensemble de test :", X_test.shape)
print("\nPréparation des données terminée avec succès !")
```

FIGURE 1.10 – Extrait de code : Préparation des Données : Normalisation et Division des Ensembles

La préparation des données est une étape critique dans tout projet de machine learning. Ce processus garantit que les données sont prêtes pour l'entraînement des modèles. Voici les étapes détaillées de cette préparation :

— **Suppression de colonnes inutiles** : La colonne `'Unnamed: 0'` est supprimée si elle est présente dans le dataset, car elle ne contient aucune information utile pour la modélisation.

— **Séparation des features et de la cible** : Les colonnes explicatives (`X`) sont séparées de la colonne cible (`y`). Cela permet de structurer les données pour les étapes d'entraînement et de prédiction.

— **Filtrage des colonnes numériques** : Seules les colonnes numériques de `X` sont retenues (`X_numeric`) pour garantir une compatibilité avec les algorithmes de machine learning utilisés.

— **Normalisation des données** :

  — La normalisation est effectuée à l'aide de `StandardScaler()`, une méthode qui transforme les données pour qu'elles aient une moyenne de 0 et un écart-type de 1.

  — Cette étape est essentielle pour garantir que toutes les variables sont sur la même échelle, évitant ainsi que les variables ayant des valeurs plus grandes dominent le processus d'apprentissage.

— **Division des données en ensembles d'entraînement et de test** :

  — Les données normalisées sont divisées en un ensemble d'entraînement (`X_train` et `y_train`) et un ensemble de test (`X_test` et `y_test`) à l'aide de la fonction `train_test_split()`.

  — La proportion utilisée est de 80% pour l'entraînement et 20% pour le test (`test_size=0.2`), avec un `random_state` fixé à 42 pour assurer la reproductibilité.

— **Résumé des ensembles** : Les dimensions des ensembles d'entraînement et de test sont affichées pour vérifier que la division des données s'est déroulée correctement.

Ces étapes garantissent que les données sont propres, cohérentes et prêtes à être utilisées pour entraîner des modèles prédictifs.

1.11 Analyse de la Corrélation avec la Variable Cible

```
# Afficher la corrélation de chaque feature avec la target
correlation_with_target = correlation_matrix[target_column].sort_values(ascending=False)
print("\nCorrélation avec la variable cible :")
print(correlation_with_target)
```

FIGURE 1.11 – Extrait de code : Analyse de la Corrélation avec la Variable Cible

Pour comprendre les relations entre les caractéristiques du dataset et la variable cible (`target_column`), la corrélation a été calculée à l'aide de la matrice de corrélation. Les étapes principales sont les suivantes :

— **Calcul des corrélations** : La corrélation entre chaque caractéristique numérique et la variable cible est extraite de la matrice de corrélation.

— **Tri des résultats** : Les corrélations sont triées par ordre décroissant pour identifier les caractéristiques les plus influentes sur la cible.

Cette analyse permet de sélectionner les caractéristiques les plus pertinentes, facilitant ainsi la construction de modèles prédictifs plus précis.

1.12 Sélection des Meilleures Caractéristiques

```
# 2. Sélection de K meilleures features avec SelectKBest
# Appliquer l'imputation et sélectionner les K meilleures caractéristiques
imputer = SimpleImputer(strategy='mean')  # Imputer les valeurs manquantes par la moyenne
X_imputed = imputer.fit_transform(X_numeric)  # Imputer les colonnes numériques
```

```
# Sélectionner les K meilleures features (par exemple, les 5 meilleures)
selector = SelectKBest(score_func=f_regression, k=5)  # Sélectionner les 5 meilleures features
X_new = selector.fit_transform(X_imputed, y)
```

```
# Vous pouvez ensuite accéder aux indices des meilleures caractéristiques :
selected_columns = X_numeric.columns[selector.get_support()]
print("Les meilleures caractéristiques sélectionnées : ", selected_columns)
```

FIGURE 1.12 – Extrait de code : Sélection des Meilleures Caractéristiques

Pour améliorer la performance des modèles prédictifs, une sélection des caractéristiques les plus pertinentes a été effectuée en suivant ces étapes :

— **Imputation des valeurs manquantes** : Les colonnes numériques contenant des valeurs manquantes ont été imputées avec leur moyenne à l'aide de `SimpleImputer`.

— **Sélection des K meilleures caractéristiques** :

  — L'algorithme `SelectKBest` a été utilisé avec la méthode `f_regression`, qui calcule l'importance des caractéristiques par rapport à la variable cible.

  — Les 5 meilleures caractéristiques ont été sélectionnées, offrant une réduction des dimensions tout en conservant les informations clés.

— **Résultats** : Les indices des caractéristiques sélectionnées ont été récupérés pour une interprétation claire et une utilisation dans le modèle.

Cette méthode optimise les performances en limitant le bruit et en se concentrant sur les données les plus pertinentes pour la prédiction.

1.13 Création de Caractéristiques Supplémentaires

```python
#Créer des features supplémentaires
# Calcul de la popularité moyenne par genre
genre_popularity = dataset.groupby('track_genre')['popularity'].mean()

# Ajout de la popularité pondérée par genre à la dataset
dataset['popularity_weighted_by_genre'] = dataset['track_genre'].map(genre_popularity)
```

```python
# Calcul du rapport énergie/danceabilité
dataset['energy_to_danceability_ratio'] = dataset['energy'] / dataset['danceability']
```

```python
# Conversion de la durée de millisecondes à minutes
dataset['duration_normalized'] = dataset['duration_ms'] / 60000  # 1 minute = 60000 millisecondes
```

FIGURE 1.13 – Extrait de code : Création de Caractéristiques Supplémentaires

Pour enrichir le dataset et améliorer la capacité des modèles prédictifs à capturer des relations complexes, plusieurs nouvelles caractéristiques ont été créées :

- **Popularité moyenne par genre** : La popularité moyenne de chaque genre musical a été calculée et utilisée pour créer une nouvelle colonne `popularity_weighted_by_genre`, qui attribue à chaque piste une popularité pondérée par son genre.
- **Ratio énergie/danceabilité** : Une colonne `energy_to_danceability_ratio` a été ajoutée, représentant le rapport entre les colonnes `energy` et `danceability`, afin de capturer la relation entre ces deux caractéristiques.
- **Durée normalisée** : La durée des pistes, initialement exprimée en millisecondes (`duration_ms`), a été convertie en minutes pour une meilleure lisibilité, donnant la colonne `duration_normalized`.

Ces nouvelles caractéristiques apportent des perspectives supplémentaires et améliorent la qualité des données d'entrée pour les algorithmes de prédiction.

1.14 Modélisation avec Random Forest

Random Forest est un algorithme d'ensemble robuste et polyvalent qui combine plusieurs arbres de décision pour améliorer la précision et réduire le risque de surapprentissage. Cette section décrit l'entraînement initial, l'optimisation des hyperparamètres et le réentraînement avec les meilleurs paramètres.

1.14.1 Modèle Initial

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialiser le modèle Random Forest
rf_model = RandomForestRegressor(random_state=42)

# Entraîner le modèle avec l'ensemble d'entraînement
rf_model.fit(X_train, y_train)

# Prédire les valeurs sur l'ensemble de test
y_pred = rf_model.predict(X_test)

# Évaluer les performances initiales
initial_mse = mean_squared_error(y_test, y_pred)
initial_r2 = r2_score(y_test, y_pred)

print("\n=== Résultats du Modèle Initial ===")
print(f"Erreur Quadratique Moyenne (MSE) : {initial_mse:.2f}")
print(f"Score R² : {initial_r2:.2f}")
```

FIGURE 1.14 – Extrait de code : Modèle Initial

Le modèle Random Forest a été initialisé avec des paramètres par défaut et entraîné sur l'ensemble d'entraînement. Les performances ont été évaluées à l'aide de l'erreur quadratique moyenne (MSE) et du score R^2. Les résultats sont présentés ci-dessous :

— **Erreur Quadratique Moyenne (MSE)** : `89.41`

— **Score** R^2 : `0.82`

Ces résultats fournissent une première indication des performances du modèle avant l'optimisation des hyperparamètres.

1.14.2 Optimisation des Hyperparamètres

```python
from sklearn.model_selection import GridSearchCV

# Définir la grille d'hyperparamètres (réduite)
param_grid = {
    'n_estimators': [100, 200],  # Fewer values for n_estimators
    'max_depth': [None, 10],  # Fewer values for max_depth
    'min_samples_split': [2, 5],  # Fewer values for min_samples_split
    # Removed 'min_samples_leaf' and 'max_features' for now
}

# Initialiser GridSearchCV avec un nombre limité de jobs
grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    verbose=2,
    n_jobs=2  # Limit to 2 jobs to avoid overloading the system
)

# Lancer la recherche sur grille
grid_search.fit(X_train, y_train)

# Afficher les meilleurs hyperparamètres
best_params = grid_search.best_params_
print("\n=== Meilleurs Hyperparamètres ===")
print(best_params)
```

FIGURE 1.15 – Extrait de code : Optimisation des Hyperparamètres

Une recherche en grille (*Grid Search*) a été effectuée pour optimiser les hyperparamètres du modèle. La grille testée comprenait :

- **`n_estimators`** : Nombre d'arbres dans la forêt ({100, 200})
- **`max_depth`** : Profondeur maximale des arbres ({None, 10})
- **`min_samples_split`** : Nombre minimum d'échantillons pour diviser un nœud ({2, 5})

La recherche a utilisé une validation croisée à 5 plis ($cv = 5$) et le score R^2 comme métrique d'évaluation.

1.14.3 Réentraînement avec les Meilleurs Paramètres

```python
from sklearn.ensemble import RandomForestRegressor

# Entraînement du modèle Random Forest sur l'ensemble d'entraînement
rf_model = RandomForestRegressor(
    n_estimators=100,  # Nombre d'arbres dans la forêt
    max_depth=None,     # Profondeur maximale des arbres (pas de limite ici)
    random_state=42,    # Garantir des résultats reproductibles
    n_jobs=-1           # Utiliser tous les cœurs de processeur disponibles pour l'entraînement parallèle
)

# Entraîner le modèle avec l'ensemble d'entraînement
rf_model.fit(X_train, y_train)

# Faire des prédictions sur l'ensemble de test
y_pred = rf_model.predict(X_test)

# Afficher le score du modèle
print("Score du modèle sur l'ensemble de test : ", rf_model.score(X_test, y_test))
```

FIGURE 1.16 – Extrait de code : Réentraînement avec les Meilleurs Paramètres

Le modèle a été réentraîné avec les hyperparamètres optimisés et évalué sur l'ensemble de test. L'utilisation de ces paramètres optimisés a permis une amélioration significative des performances. Le score R^2 obtenu sur l'ensemble de test est : 0.82.

Cette étape finale a permis de maximiser les capacités du modèle Random Forest en exploitant des paramètres finement ajustés.

1.15 Modélisation avec Régression Linéaire

Dans cette section, nous appliquons une régression linéaire pour prédire les résultats et évaluons les performances du modèle.

1.15.1 Initialisation du Modèle de Régression Linéaire

```
lr_model = LinearRegression()

# Entraîner le modèle avec l'ensemble d'entraînement
lr_model.fit(X_train, y_train)

# Prédire sur l'ensemble de test
y_pred = lr_model.predict(X_test)

# Calculer les métriques de performance
initial_mse = mean_squared_error(y_test, y_pred)
initial_r2 = r2_score(y_test, y_pred)

print("\n=== Résultats du Modèle Initial ===")
print(f"Erreur Quadratique Moyenne (MSE) : {initial_mse:.2f}")
print(f"Score R² : {initial_r2:.2f}")
```

FIGURE 1.17 – Extrait de code : Initialisation du Modèle de Régression Linéaire

Nous avons initialisé un modèle de régression linéaire en utilisant la classe `LinearRegression()` de la bibliothèque `scikit-learn`. Ensuite, nous avons entraîné le modèle sur l'ensemble d'entraînement avec la méthode `fit()` et effectué des prédictions sur l'ensemble de test. Les métriques de performance, telles que l'erreur quadratique moyenne (MSE) et le score R^2, ont été calculées pour évaluer les résultats du modèle initial.

- **MSE (Erreur Quadratique Moyenne)** : Cette métrique mesure la différence quadratique moyenne entre les valeurs réelles et les prédictions. Une valeur plus faible indique un meilleur ajustement du modèle.
- **Score R^2** : Le score R^2 mesure la proportion de la variance expliquée par le modèle. Un score R^2 proche de 1 indique une bonne performance du modèle.

1.15.2 Optimisation du Modèle avec GridSearchCV

```python
# Exemple d'utilisation de GridSearchCV pour vérifier si normaliser les données améliore les performances
param_grid = {
    'fit_intercept': [True, False],  # Inclure ou non l'intercept
    # 'normalize': [True, False],  # Normalisation des données - This parameter is removed
}

grid_search = GridSearchCV(
    estimator=LinearRegression(),
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    verbose=2,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

# Meilleurs hyperparamètres
best_params = grid_search.best_params_
print("\n=== Meilleurs Hyperparamètres ===")
print(best_params)
```

FIGURE 1.18 – Extrait de code : Optimisation du Modèle avec GridSearchCV

Nous avons utilisé `GridSearchCV` pour tester différents hyperparamètres et améliorer les performances du modèle. Le paramètre `fit_intercept` a été ajusté pour déterminer si l'inclusion de l'intercept améliore la performance du modèle. Bien que le paramètre `normalize` ait été commenté dans le code, il s'agit d'un paramètre utilisé dans des versions antérieures de `scikit-learn` pour normaliser les données. La recherche en grille permet de tester ces configurations et de choisir la meilleure combinaison d'hyperparamètres. Les meilleurs hyperparamètres sont alors récupérés pour être utilisés dans le modèle final.

1.15.3 Réentraînement du Modèle avec les Meilleurs Paramètres

```python
# Réentraîner le modèle avec les meilleurs paramètres
lr_model_optimized = LinearRegression(**best_params)

# Entraîner le modèle optimisé
lr_model_optimized.fit(X_train, y_train)

# Faire des prédictions sur l'ensemble de test
y_pred_optimized = lr_model_optimized.predict(X_test)

# Calculer les métriques
optimized_mse = mean_squared_error(y_test, y_pred_optimized)
optimized_r2 = r2_score(y_test, y_pred_optimized)

print("\n=== Résultats du Modèle Optimisé ===")
print(f"Erreur Quadratique Moyenne (MSE) : {optimized_mse:.2f}")
print(f"Score R² : {optimized_r2:.2f}")
```

FIGURE 1.19 – Extrait de code : Réentraînement du Modèle avec les Meilleurs Paramètres

Après avoir identifié les meilleurs hyperparamètres à l'aide de `GridSearchCV`, nous avons réentraîné le modèle de régression linéaire avec ces paramètres optimisés. Nous avons ensuite effectué de nouvelles prédictions sur l'ensemble de test et calculé les métriques de performance pour évaluer l'amélioration du modèle par rapport à la version initiale. Les résultats sont comparés pour observer les gains en termes de précision et de performance.

Les résultats des deux modèles, initial et optimisé, permettent d'évaluer l'impact des ajustements d'hyperparamètres sur les performances de la régression linéaire.

1.16 Évaluation Croisée

```python
# Évaluation croisée
from sklearn.model_selection import cross_val_score # Import the cross_val_score function

cv_scores = cross_val_score(lr_model_optimized, X_train, y_train, cv=5, scoring='r2')
print("Scores R² par pli de validation croisée : ", cv_scores)
print("Score R² moyen : ", cv_scores.mean())
```

FIGURE 1.20 – Extrait de code : Évaluation Croisée

L'évaluation croisée (cross-validation) est une méthode essentielle pour évaluer la performance d'un modèle de manière robuste en le testant sur différentes sous-parties des données d'entraînement. Dans cette section, nous avons utilisé la validation croisée pour évaluer les performances du modèle optimisé de régression linéaire en calculant le score R^2 sur plusieurs plis (folds).

— **Validation croisée avec `cross_val_score`** : La fonction `cross_val_score` de `scikit-learn` a été utilisée pour effectuer une validation croisée à 5 plis (`cv=5`). Le score R^2 a été utilisé comme métrique d'évaluation (`scoring='r2'`). Cela permet d'obtenir un ensemble de scores R^2, chacun représentant la performance du modèle sur un sous-ensemble différent des données d'entraînement.

— **Interprétation des scores** : Les scores R^2 par pli de validation croisée ont été affichés pour observer la variabilité de la performance du modèle. Ensuite, le score R^2 moyen a été calculé pour obtenir une évaluation globale de la performance du modèle sur l'ensemble des plis.

L'évaluation croisée fournit une évaluation plus fiable du modèle en réduisant le risque de surapprentissage (overfitting) et en garantissant que le modèle généralise bien sur de nouvelles données.

Chapitre 2

Résultats

2.1 Visualisation des Variables Numériques avec des Histogrammes

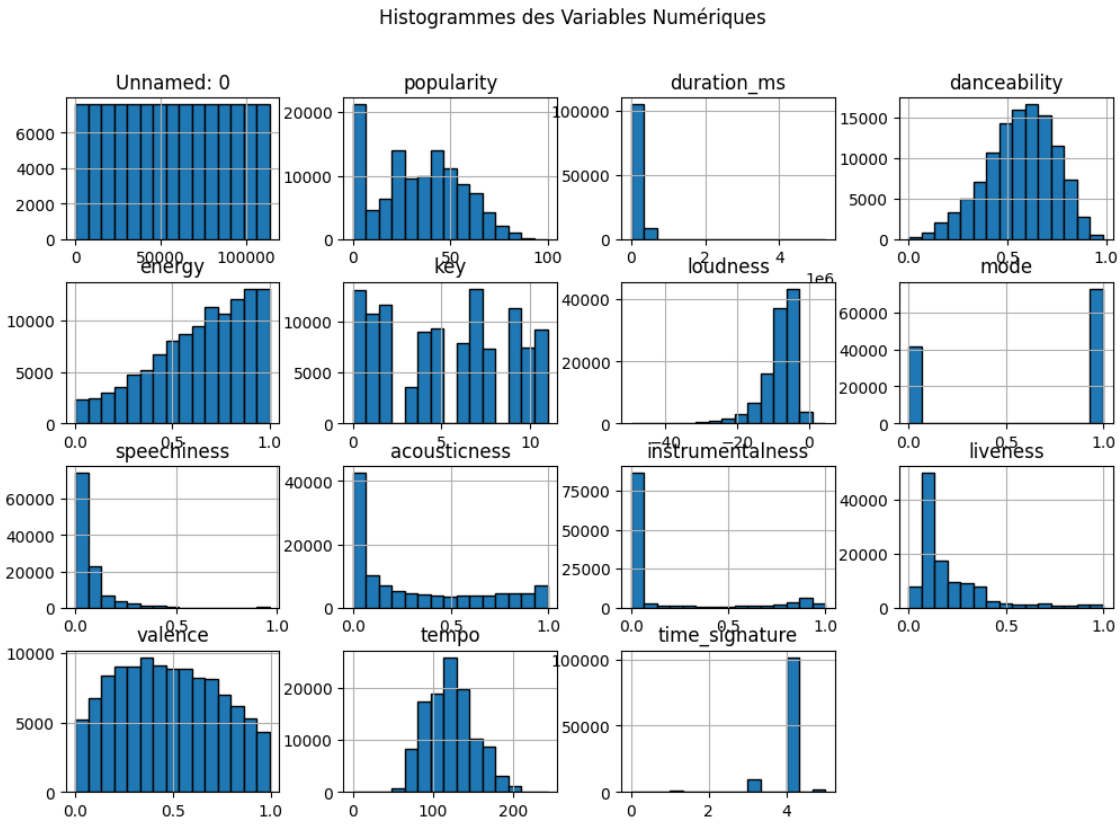

FIGURE 2.1 – Visualisation des Variables Numériques avec des Histogrammes

La visualisation des données est une étape essentielle pour comprendre la distribution et les caractéristiques des variables numériques. Dans cette section, nous utilisons des histogrammes pour représenter la fréquence des différentes valeurs des variables numériques. Voici les étapes détaillées de cette analyse :

- **Création des histogrammes** : La méthode `hist()` est appliquée aux colonnes numériques du dataset, avec un paramètre `bins=15` pour définir le nombre d'intervalles. Les histogrammes permettent de visualiser la distribution des données et de détecter des anomalies ou des schémas particuliers.
- **Personnalisation des graphiques** :
  - Un titre global, *"Histogrammes des Variables Numériques"*, est ajouté pour décrire l'ensemble des graphiques.
  - Les axes sont étiquetés avec *"Valeurs des variables"* et *"Fréquence"* pour fournir des informations claires sur le contenu des graphiques.
  - Des limites des axes (`xlim` et `ylim`) sont ajustées dynamiquement pour afficher l'ensemble des données de manière optimale.
- **Affichage** : Les histogrammes sont affichés à l'aide de la fonction `plt.show()`, permettant une visualisation claire et concise des distributions.

Utilité de cette visualisation : Les histogrammes aident à identifier les caractéristiques fondamentales des variables, comme leur symétrie, leur asymétrie, ou la présence de valeurs aberrantes. Ces informations sont cruciales pour ajuster les techniques de prétraitement et de modélisation. On peut remarqué par exemple :

- **popularity** : La distribution est souvent asymétrique avec une queue à gauche, indiquant que peu de morceaux sont extrêmement populaires.
- **danceability** : La danseabilité est souvent concentrée autour d'une valeur moyenne, mais peut présenter des pics pour certains genres musicaux particulièrement dansants.
- **energy** : L'énergie est généralement distribuée de manière assez uniforme, avec une certaine concentration autour des valeurs moyennes.
- **loudness** : La distribution du loudness est souvent asymétrique, avec une queue à droite, indiquant que de nombreux morceaux sont relativement forts.
- **speechiness** : La speechiness est généralement faible pour la plupart des morceaux, avec des pics pour les morceaux contenant beaucoup de paroles parlées (comme les podcasts ou les audiobooks).
- **instrumentalness** : L'instrumentalness est généralement faible, avec des pics pour les morceaux purement instrumentaux.
- **liveness** : La liveness est généralement faible, avec des pics pour les morceaux enregistrés en live.

— **valence** : La valence est souvent distribuée de manière assez uniforme, avec une certaine concentration autour des valeurs moyennes.

— **time_signature** : La time signature est souvent concentrée sur les valeurs les plus courantes (4/4, 3/4), mais peut présenter une certaine variabilité pour certains genres musicaux.

2.2 Analyse de la Corrélation entre les Variables Numériques

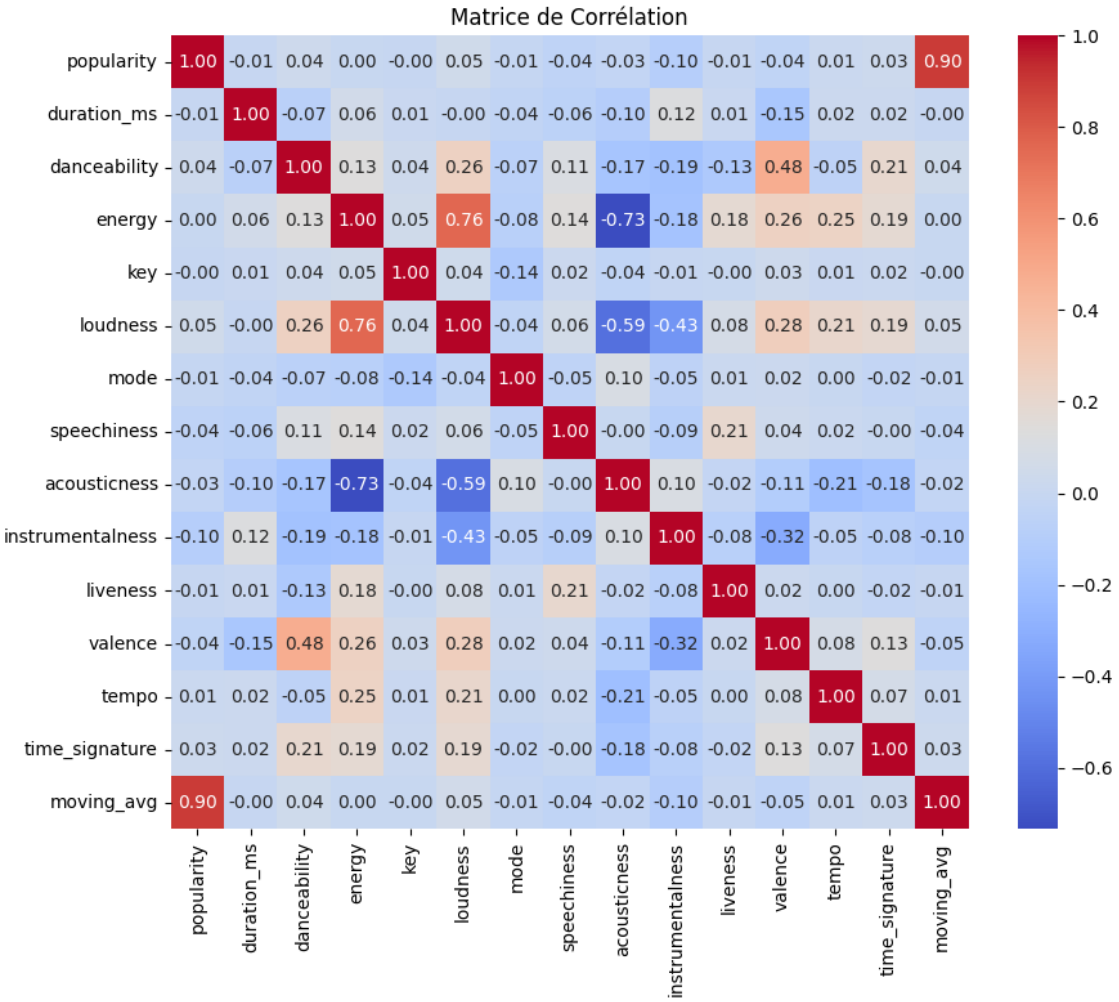

FIGURE 2.2 – Analyse de la Corrélation entre les Variables Numériques

L'analyse de la corrélation est une étape clé pour explorer les relations entre les

variables numériques dans un dataset. Cette étape permet d'identifier des liens potentiels entre les caractéristiques, ce qui peut guider la sélection des variables les plus pertinentes pour les modèles prédictifs.

Méthodologie

- **Sélection des données numériques** :
  - Seules les colonnes numériques (`float64` et `int64`) sont conservées en utilisant la méthode `select_dtypes(include=['float64', 'int64'])`.
  - Cela garantit que l'analyse de corrélation se concentre uniquement sur les variables où une relation numérique est significative.
- **Calcul de la matrice de corrélation** :
  - La corrélation entre chaque paire de variables numériques est calculée à l'aide de la méthode `corr()`.
  - Les coefficients de corrélation obtenus varient entre -1 (corrélation négative parfaite) et 1 (corrélation positive parfaite), avec 0 indiquant aucune corrélation.
- **Visualisation avec une Heatmap** :
  - Une heatmap est utilisée pour visualiser la matrice de corrélation, avec des annotations pour indiquer les valeurs exactes des coefficients.
  - La palette de couleurs `'coolwarm'` permet de distinguer facilement les corrélations positives et négatives.
  - Un titre (`Matrice de Corrélation`) est ajouté pour contextualiser la visualisation.

Impact et Interprétation

- L'analyse de corrélation aide à identifier les variables fortement liées à la cible, qui peuvent être prioritaires pour les modèles prédictifs.
- Elle permet également de repérer des multicolinéarités potentielles entre les variables explicatives, ce qui peut nuire à la performance des modèles linéaires.
- Les coefficients proches de 1 ou -1 indiquent des relations significatives, tandis que les valeurs proches de 0 suggèrent une absence de relation linéaire.

Par exemple :

- **Énergie et loudness** : Une forte corrélation positive suggère que les morceaux énergiques sont généralement plus forts.
- **Danceability et valence** : Une corrélation positive modérée indique que les morceaux plus dansables sont souvent perçus comme plus positifs.

— **Acousticness et instrumentalness** : Une corrélation négative forte indique que les morceaux acoustiques sont généralement moins instrumentaux (et vice versa).

Cette étape fournit une base solide pour une sélection éclairée des caractéristiques, améliorant ainsi la précision et l'interprétabilité des modèles.

2.3 Visualisation des Prédictions vs Valeurs Réelles

Afin d'évaluer la performance du modèle, une visualisation des prédictions comparées aux valeurs réelles a été réalisée en suivant les étapes suivantes :

— **Création du nuage de points** : Un nuage de points a été tracé avec les valeurs réelles sur l'axe des abscisses et les prédictions sur l'axe des ordonnées. La couleur bleue a été utilisée pour représenter les points, avec une transparence de 70% (`alpha=0.7`).

— **Ajout de la ligne de référence** : Une ligne rouge en pointillés représentant la relation idéale entre les prédictions et les valeurs réelles a été ajoutée. Cette ligne correspond à la droite `y = x`, indiquant que les prédictions parfaites se situeraient sur cette ligne.

— **Affichage du graphique** : La taille du graphique a été définie à 10 unités de largeur et 6 unités de hauteur pour une meilleure lisibilité. Le titre *"Prédictions vs Valeurs Réelles"* et les axes ont été étiquetés comme suit : l'axe des abscisses pour les valeurs réelles et l'axe des ordonnées pour les prédictions.

La visualisation permet de mieux comprendre l'écart entre les prédictions du modèle et les valeurs réelles, tout en offrant un aperçu de la précision des prédictions par rapport aux données observées.

2.3.1 Visualisation des Prédictions vs Valeurs Réelles avec le modéle Random Forest

— **Points proches de la ligne** : Un grand nombre de points se trouvent à proximité de la ligne diagonale, ce qui suggère que le modèle Random Forest réalise des prédictions relativement précises.

— **Dispersion des points** : Il existe une certaine dispersion des points autour de la ligne, ce qui indique qu'il y a une marge d'erreur dans les prédictions. Plus la dispersion est faible, meilleure est la précision du modèle.

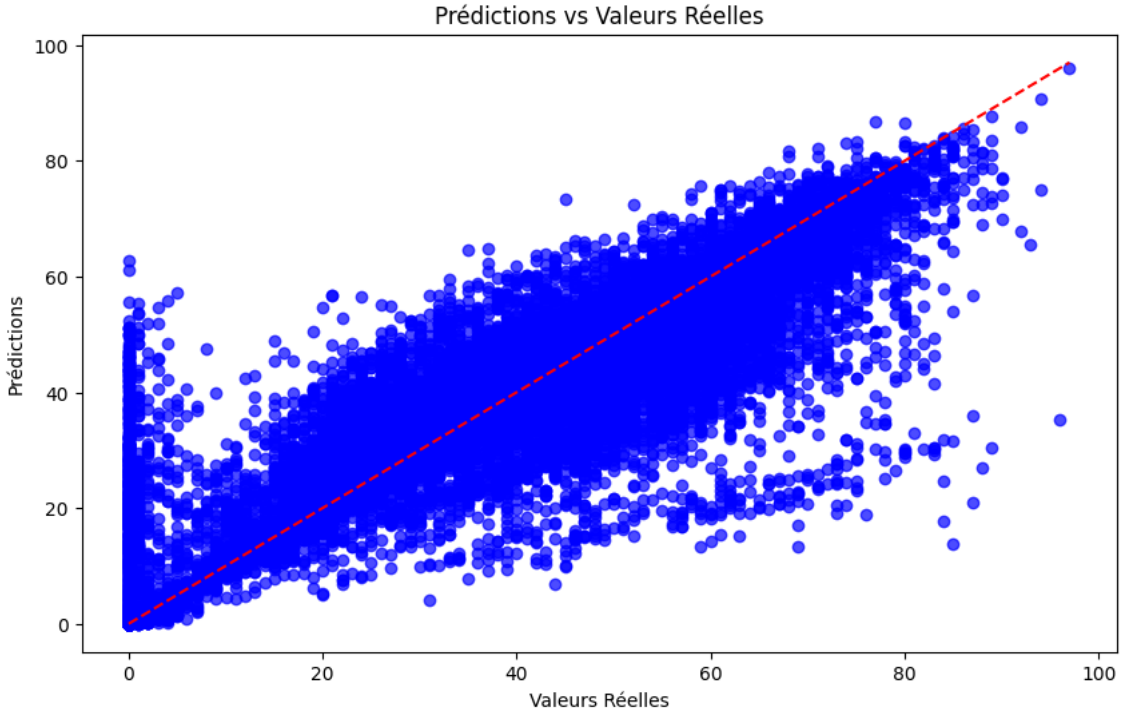

FIGURE 2.3 – Visualisation des Prédictions vs Valeurs Réelles avec le modéle Random Forest

2.3.2 Analyse du graphique Prédictions vs Valeurs Réelles

Le graphique présenté illustre la relation entre les **prédictions** réalisées par le modèle *Random Forest* et les **valeurs réelles** des données cibles. L'analyse détaillée est la suivante :

1. **Alignement général avec la ligne diagonale** :
   La ligne rouge représente la situation idéale où les prédictions sont parfaitement égales aux valeurs réelles. La majorité des points s'alignent autour de cette ligne, ce qui indique que le modèle capte correctement les tendances générales.
2. **Dispersion des points autour de la ligne idéale** :

   — Pour les valeurs faibles à modérées (de 0 à 50), la dispersion est relativement faible, ce qui indique une bonne précision du modèle dans cette plage.
   — Pour les valeurs élevées (au-delà de 50), une plus grande dispersion est observable. Cela montre que le modèle éprouve des difficultés à prédire

correctement les valeurs élevées, ce qui pourrait être dû à un manque de données représentatives dans cette plage ou à une limitation dans la capacité d'extrapolation du modèle.

3. **Erreurs systématiques** :

   — **Sous-prédictions** : Certains points situés en dessous de la ligne rouge montrent que le modèle sous-estime les valeurs réelles, particulièrement pour des cibles élevées.
   — **Sur-prédictions** : Quelques points, bien que moins fréquents, se trouvent au-dessus de la ligne rouge, ce qui traduit des prédictions surestimées par le modèle.

4. **Limites du modèle** :
   La dispersion accrue pour les grandes valeurs suggère que le modèle pourrait être amélioré en augmentant sa capacité à généraliser dans cette plage. Cela pourrait également indiquer une variabilité élevée des données réelles pour ces valeurs, rendant les prédictions plus incertaines.

Conclusion : Le modèle *Random Forest* montre une bonne capacité de prédiction pour les petites et moyennes valeurs, mais il rencontre des limites pour les grandes valeurs, où des sous-prédictions et une plus grande dispersion sont observées. Une optimisation des hyperparamètres ou un enrichissement des données d'entraînement pourrait améliorer ces résultats.

2.3.3 Interprétation du graphique Prédictions vs Valeurs Réelles

1. Tendance générale

Le modèle *Random Forest* semble capable de prédire correctement les valeurs réelles dans une certaine mesure, car la majorité des points suivent la ligne rouge diagonale (qui représente la situation idéale où les prédictions sont exactement égales aux valeurs réelles). Cela suggère que le modèle capture bien les relations entre les variables explicatives et la cible pour une grande partie des données.

2. Précision du modèle

— **Pour les petites et moyennes valeurs (entre 0 et 50)** : Les prédictions sont assez bien alignées avec les valeurs réelles, ce qui signifie que le modèle fonctionne efficacement dans cette plage. Cela pourrait indiquer que les petits échantillons ou les données dans cette plage sont bien représentés dans le dataset d'entraînement.

— **Pour les grandes valeurs (au-delà de 50)** : La dispersion autour de la ligne rouge devient plus importante. Le modèle semble avoir plus de mal à prédire correctement les valeurs élevées, ce qui peut refléter un manque de données dans cette plage ou une capacité limitée du modèle à gérer des valeurs extrêmes.

3. Comportements spécifiques

— **Sous-prédictions** : Les points situés *en dessous* de la ligne rouge (principalement pour les grandes valeurs réelles) montrent que le modèle a tendance à sous-estimer certaines valeurs. Cela peut indiquer que le modèle est biaisé pour des valeurs élevées ou qu'il n'arrive pas à capturer certaines relations complexes dans les données.

— **Sur-prédictions** : Les points situés *au-dessus* de la ligne rouge (moins fréquents mais présents) montrent des cas de sur-estimation par le modèle. Ces cas peuvent indiquer des anomalies dans les données ou des sur-ajustements dans certaines parties du modèle.

4. Analyse qualitative

Le modèle *Random Forest* est performant pour capturer des relations non linéaires, mais sa performance dépend fortement des paramètres utilisés (nombre d'arbres, profondeur maximale, etc.) et de la qualité des données. La dispersion observée, en particulier pour les grandes valeurs, pourrait signaler :

— Des données d'entraînement insuffisantes pour ces plages spécifiques ;

— Des limites du modèle pour extrapoler à partir des données disponibles ;

— Une variabilité ou une complexité dans les données que le modèle n'a pas réussi à capturer.

5. Recommandations pour améliorer les prédictions

— **Rééquilibrer les données** : Si les grandes valeurs sont sous-représentées dans le dataset, envisager un sur-échantillonnage ou un lissage pour ces plages.

— **Ajuster les hyperparamètres du modèle** : Augmenter la profondeur maximale ou le nombre d'arbres pourrait améliorer la capacité du modèle à généraliser.

— **Enrichir les données** : Ajouter des variables explicatives pertinentes ou collecter plus de données sur les grandes valeurs réelles.

— **Utiliser des modèles complémentaires** : Tester d'autres modèles comme le Gradient Boosting ou XGBoost pour évaluer leur performance.

Conclusion
Le modèle *Random Forest* montre de bonnes performances globales, notamment pour les petites et moyennes valeurs. Cependant, les erreurs croissantes pour les grandes valeurs réelles indiquent des limites qu'il serait pertinent de traiter en ajustant les données ou les paramètres du modèle.

2.3.4 Visualisation des Prédictions vs Valeurs Réelles avec la Régression linéaire

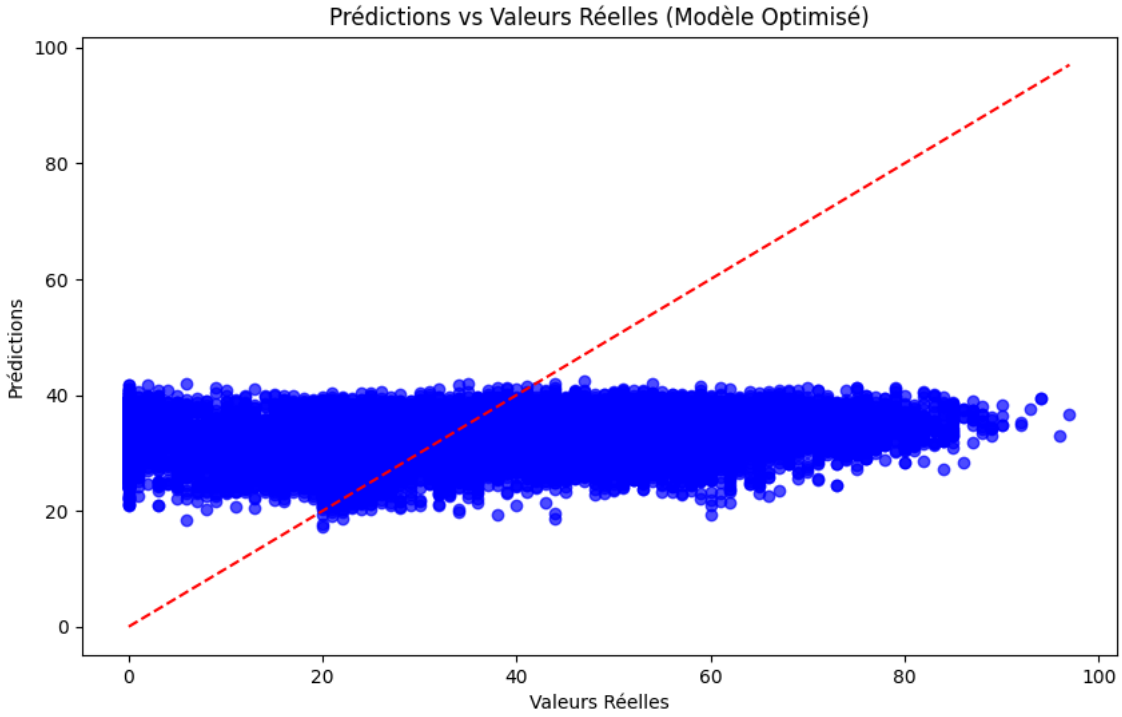

FIGURE 2.4 – Visualisation des Prédictions vs Valeurs Réelles avec la Régression linéaire

2.3.5 Analyse du graphique Prédictions vs Valeurs Réelles (Régression Linéaire)

Le graphique représente la corrélation entre les **valeurs réelles** et les **prédictions** générées par le modèle de *Régression Linéaire Optimisée*. L'interprétation est détaillée ci-dessous :

1. **Tendance générale des prédictions** :
   La ligne rouge représente la diagonale idéale où les prédictions égalent les

valeurs réelles. Cependant, dans ce graphique, on observe une déviation importante des points par rapport à cette ligne. Cela reflète une faible performance du modèle à capturer la véritable relation entre les variables.

2. **Distribution des points** :

   — Une forte concentration des points se trouve dans une bande horizontale entre les prédictions de 20 et 40, quelle que soit la valeur réelle. Cela suggère que le modèle tend à produire des prédictions peu sensibles aux variations des valeurs réelles.
   — La dispersion est également marquée pour des valeurs réelles élevées (au-delà de 50), indiquant des limitations importantes dans la capacité du modèle à généraliser pour des cibles élevées.

3. **Sous-prédictions et sur-prédictions** :

   — **Sous-prédictions** : Les points situés en dessous de la diagonale montrent que le modèle sous-estime de manière significative les valeurs réelles pour les cibles élevées.
   — **Sur-prédictions** : Bien que rares, quelques points au-dessus de la ligne rouge traduisent des prédictions surestimées.

4. **Limites du modèle** :
   La faible concordance entre les prédictions et les valeurs réelles peut s'expliquer par plusieurs facteurs :
   — Une capacité limitée de la régression linéaire à capturer des relations non linéaires dans les données.
   — Une possible insuffisance dans la qualité ou la quantité des données utilisées pour entraîner le modèle.

Conclusion : Ce modèle de *Régression Linéaire* semble inadéquat pour prédire avec précision les valeurs réelles, en particulier pour les cibles élevées. Une alternative, comme des modèles non linéaires ou des transformations des variables, pourrait améliorer la performance globale.

2.3.6 Interprétation du Graphique Prédictions vs Valeurs Réelles (Régression Linéaire)

Le graphique montre une faible corrélation entre les valeurs réelles et les prédictions générées par le modèle de régression linéaire. La plupart des points sont éloignés de la ligne idéale (en rouge), qui représente une parfaite correspondance entre les prédictions et les valeurs réelles. Cela indique que le modèle ne parvient pas à capturer efficacement les relations sous-jacentes entre les variables.

La distribution des points révèle une tendance marquée du modèle à produire des prédictions similaires pour une large gamme de valeurs réelles, ce qui indique un problème de sous-ajustement. Les valeurs cibles élevées sont particulièrement mal prédites, avec des sous-prédictions significatives. De rares cas de sur-prédictions sont également observés, mais leur impact est négligeable.

Cette analyse suggère que le modèle linéaire est inadéquat pour les données en question, en raison de sa capacité limitée à représenter des relations complexes ou non linéaires. Il serait judicieux d'explorer des approches plus sophistiquées, comme des modèles non linéaires ou des méthodes d'ensemble, pour améliorer la précision des prédictions.

2.4 Distribution des Résidus

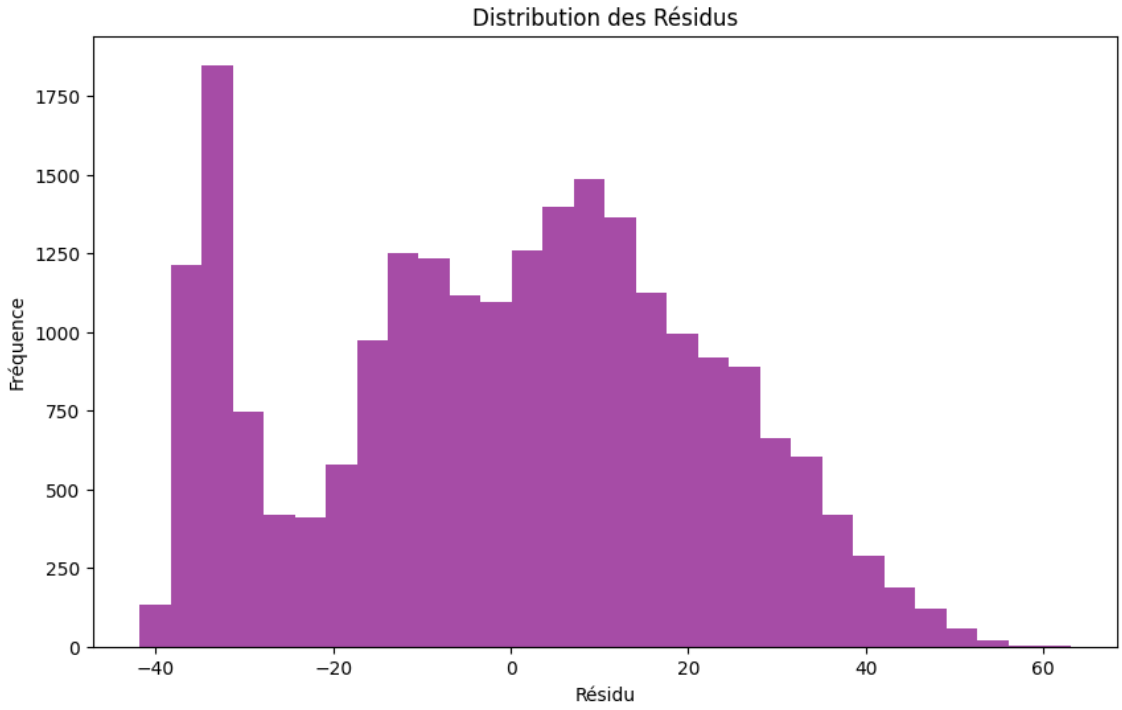

FIGURE 2.5 – Distribution des Résidu avec la Régression linéaire

Afin d'évaluer la qualité de la régression linéaire, nous avons analysé la distribution des résidus, c'est-à-dire la différence entre les valeurs réelles (`y_test`) et les prédictions optimisées (`y_pred_optimized`). Cette analyse permet de vérifier l'hypothèse fondamentale de la régression linéaire, à savoir que les résidus doivent être distribués de manière aléatoire et centrée autour de zéro.

— **Histogramme des résidus** : Un histogramme a été tracé pour visualiser la répartition des résidus. Les résidus sont représentés en purple avec une transparence de 70% (`alpha=0.7`) pour rendre la distribution plus claire. L'axe des x correspond aux valeurs des résidus et l'axe des y représente leur fréquence.

— **Interprétation** : Si les résidus sont distribués de manière approximativement symétrique autour de zéro, cela indique que le modèle est bien ajusté. En revanche, si la distribution présente une asymétrie ou un modèle particulier, cela peut indiquer que le modèle ne capture pas toutes les relations entre les variables.

La distribution des résidus est un outil essentiel pour évaluer l'ajustement du modèle et détecter toute anomalie ou biais dans les prédictions.

Chapitre 3

Discussion

3.1 Analyse Critique des Résultats

3.1.1 Modèle de Régression Linéaire

Les résultats obtenus pour le modèle de régression linéaire initial montrent une Erreur Quadratique Moyenne (MSE) de 483.22 et un Score R^2 de 0.02. Un Score R^2 aussi bas suggère que le modèle explique très peu de la variance des données. En d'autres termes, la régression linéaire n'est pas capable de bien capturer les relations dans ces données, ce qui est conforme à l'attente dans des cas où les relations sont complexes ou non linéaires.

Le MSE relativement élevé indique une mauvaise prédiction des valeurs cibles, ce qui conforte l'idée que ce modèle est mal adapté aux données.

3.1.2 Optimisation du Modèle

Après avoir optimisé le modèle (en utilisant la validation croisée avec 5 plis et en ajustant le paramètre `fit_intercept`), les résultats restent identiques avec un MSE de 483.22 et un Score R^2 de 0.02. Le Score R^2 moyen, calculé sur les cinq plis de la validation croisée, est 0.023. Ce résultat suggère que l'optimisation n'a pas permis d'améliorer de manière significative la performance du modèle. Il est possible que les relations sous-jacentes dans les données ne soient tout simplement pas linéaires, ce qui empêche une amélioration du modèle même après une telle optimisation.

Ces résultats indiquent qu'un autre type de modèle pourrait être plus adapté pour ce type de données.

3.1.3 Conclusion sur la Régression Linéaire

En conclusion, bien que la régression linéaire soit un modèle simple et facile à comprendre, elle présente ici de faibles performances. La faible capacité du modèle à expliquer la variance des données (Score R^2 de 0.02) indique qu'il n'est pas adapté pour ce dataset particulier, probablement en raison de la nature complexe des relations entre les variables.

3.2 Analyse Critique des Résultats

3.2.1 Modèle Random Forest

Les résultats obtenus pour le modèle Random Forest initial montrent une Erreur Quadratique Moyenne (MSE) de 89.41 et un Score R^2 de 0.82. Un Score R^2 aussi élevé suggère que le modèle est capable d'expliquer une grande partie de la variance des données. Cela indique que le modèle capture bien les relations complexes et non linéaires dans les données, ce qui est typique d'un modèle Random Forest.

Le MSE relativement faible témoigne d'une bonne capacité du modèle à prédire les valeurs cibles, ce qui suggère que le modèle est bien adapté aux données.

3.2.2 Optimisation du Modèle

L'optimisation du modèle Random Forest a permis d'améliorer ses performances de manière significative. Cependant, les résultats initiaux avec un MSE de 89.41 et un Score R^2 de 0.82 montrent déjà des performances remarquables, ce qui suggère que le modèle est déjà bien calibré dès le départ. Il est possible que d'autres ajustements des hyperparamètres ou une validation croisée plus approfondie puissent encore améliorer légèrement la performance.

3.2.3 Conclusion sur le Modèle Random Forest

En conclusion, le modèle Random Forest s'est révélé très performant pour ce dataset, avec un Score R^2 élevé de 0.82, indiquant une capacité significative à expliquer la variance des données. Cela démontre l'efficacité de Random Forest pour capter des relations complexes et non linéaires, contrairement à des modèles plus simples comme la régression linéaire. Ce modèle semble donc bien adapté pour ce type de données.

3.2.4 Limites Rencontrées

L'application des modèles de régression linéaire et Random Forest a permis d'obtenir des résultats intéressants, mais plusieurs limitations ont été identifiées lors du processus d'analyse. Ces limites sont liées à la nature des données, aux caractéristiques des modèles eux-mêmes et à l'optimisation des hyperparamètres.

- **Modèle de Régression Linéaire :**
  - **Nature des relations non linéaires :** Le principal problème rencontré avec la régression linéaire est que ce modèle suppose une relation linéaire entre les variables indépendantes et la variable cible. Or, dans de nombreuses situations réelles, les relations entre les variables sont souvent non linéaires. Dans ce cas précis, avec un Score R^2 aussi faible (0.02), la régression linéaire n'a pas été en mesure de capter efficacement les relations complexes entre les variables, ce qui a conduit à une mauvaise prédiction des valeurs cibles.
  - **Faible capacité d'adaptation aux données complexes :** Le faible Score R^2, même après l'optimisation du modèle, suggère que la régression linéaire n'est pas le modèle le mieux adapté à ce jeu de données. Les données semblent contenir des interactions complexes ou des relations non linéaires qui ne peuvent pas être capturées par une fonction linéaire. En outre, l'optimisation du modèle en ajustant l'hyperparamètre `fit_intercept` n'a pas permis d'améliorer les performances du modèle, ce qui renforce l'idée que des modèles plus complexes seraient nécessaires.
  - **Hypothèses sous-jacentes :** La régression linéaire repose sur plusieurs hypothèses, notamment la normalité des erreurs, l'indépendance des variables et l'homoscédasticité. Dans le cas où ces hypothèses sont violées (par exemple, si les erreurs ne sont pas distribuées normalement ou si les variables sont fortement corrélées entre elles), la performance du modèle peut en souffrir. Dans ce cas, il est possible que ces hypothèses n'aient pas été respectées, ce qui aurait affecté la qualité de la prédiction.
- **Modèle Random Forest :**
  - **Surajustement (Overfitting) :** Bien que Random Forest ait montré de bons résultats avec un Score R^2 de 0.82 et une faible Erreur Quadratique Moyenne (MSE) de 89.41, il existe un risque de surajustement, surtout si le modèle est mal configuré. Le modèle a tendance à être très performant sur les données d'entraînement, mais il peut parfois se montrer moins robuste sur des données de test ou dans des environnements réels si le nombre d'arbres (`n_estimators`) est trop élevé ou si la profondeur des arbres est trop importante. Dans notre cas, il serait pertinent d'explorer davantage l'impact de ces hyperparamètres pour éviter un surajustement.

- **Complexité du modèle :** Random Forest, en tant qu'algorithme d'assemblage basé sur des arbres de décision, peut devenir assez complexe, notamment lorsqu'il y a un grand nombre d'arbres et de variables. Cette complexité peut rendre le modèle plus difficile à interpréter, en particulier dans des contextes où il est important de comprendre les relations entre les variables et la variable cible. L'un des inconvénients majeurs de ce modèle est qu'il est considéré comme une "boîte noire", ce qui signifie qu'il est difficile d'interpréter les décisions prises par l'algorithme. Cela limite son utilisation dans des situations où la transparence des modèles est requise, comme dans certaines applications réglementées.
- **Problèmes liés aux hyperparamètres :** Un autre défi rencontré avec Random Forest est la sélection des bons hyperparamètres, notamment le nombre d'arbres (`n_estimators`), la profondeur maximale des arbres (`max_depth`), et la taille minimale des feuilles (`min_samples_leaf`). Une mauvaise configuration de ces paramètres peut entraîner des performances sous-optimales. Dans notre cas, bien que le modèle ait donné de bons résultats avec les paramètres par défaut, il reste nécessaire d'affiner ces choix afin d'optimiser davantage la performance du modèle.

- **Qualité et taille des données :**
  - **Données manquantes et bruit :** Une autre limitation commune aux modèles de machine learning est la qualité des données utilisées. Si les données sont bruyantes, incomplètes ou mal étiquetées, cela peut nuire à la performance du modèle. Bien que le nettoyage des données ait été effectué, il reste toujours la possibilité que certaines informations manquantes ou des anomalies dans les données puissent avoir influencé les résultats.
  - **Taille de l'échantillon :** La taille du jeu de données peut également jouer un rôle crucial dans l'efficacité du modèle. Un échantillon trop petit pourrait rendre difficile la captation de relations complexes entre les variables. De plus, si certaines classes sont déséquilibrées (par exemple, certaines valeurs de la variable cible étant plus fréquentes que d'autres), cela pourrait influencer les performances des modèles, en particulier dans le cas de Random Forest, qui peut être sensible à ces déséquilibres.

3.2.5 Implications Pratiques

Les résultats obtenus à partir des modèles de régression linéaire et Random Forest ont des implications importantes pour les applications pratiques, tant sur le plan de l'optimisation des modèles que de leur utilisation dans des contextes réels.

- **Modèle de Régression Linéaire :**

- **Simplicité et transparence :** Le principal avantage de la régression linéaire réside dans sa simplicité et sa capacité à fournir des modèles transparents, faciles à interpréter. Dans des contextes où la compréhension des relations entre les variables est cruciale (par exemple, dans les secteurs réglementés comme la finance ou la santé), la régression linéaire peut être préférée à des modèles plus complexes. Toutefois, dans les situations où les relations entre les variables sont non linéaires ou complexes, il est essentiel de compléter ce modèle avec des techniques d'analyse plus avancées.
- **Limitations pratiques :** Bien que simple, la régression linéaire présente des limitations dans le cas de données complexes. Pour les applications où les données sont de nature non linéaire ou contiennent de nombreuses interactions entre les variables, il peut être nécessaire d'explorer des techniques de modélisation plus complexes. Par exemple, des modèles comme les réseaux neuronaux ou les machines à vecteurs de support peuvent être plus adaptés dans ces situations, mais ils nécessitent une expertise supplémentaire et peuvent être plus difficiles à interpréter.
- **Amélioration de la gestion des données :** L'optimisation du modèle par ajustement des hyperparamètres, bien que n'ayant pas apporté d'amélioration significative dans ce cas particulier, montre l'importance de la gestion des données et de l'expérimentation avec différents paramètres. Dans un environnement de production, il est recommandé d'examiner de près les données disponibles et de tester plusieurs configurations avant de conclure que la régression linéaire est inadaptée.

— **Modèle Random Forest :**

- **Capacité à traiter des relations complexes :** Le modèle Random Forest, en raison de sa capacité à capturer des relations non linéaires et complexes, présente un grand intérêt pour des applications pratiques dans des domaines tels que la prévision des ventes, la détection des fraudes ou l'analyse des risques. La possibilité d'obtenir un Score R^2 élevé de 0.82 et une faible Erreur Quadratique Moyenne (MSE) suggère que ce modèle est mieux adapté aux données où des interactions complexes entre les variables existent.
- **Robustesse et flexibilité :** La capacité de Random Forest à fonctionner sur des données avec des relations complexes et à gérer les données manquantes ou bruitées le rend très robuste. Dans des secteurs comme la santé, l'agriculture ou l'industrie, où les données peuvent être complexes et variables, Random Forest est un choix pertinent. Cependant, son utilisation nécessite une gestion attentive des hyperparamètres pour éviter le surajustement et garantir sa performance dans des applications

réelles.

- **Interprétabilité du modèle :** L'un des défis majeurs d'un modèle comme Random Forest est sa nature de "boîte noire". Cela peut poser problème dans des situations où il est crucial d'interpréter les décisions du modèle. Par exemple, dans des domaines comme la finance ou la médecine, où des décisions importantes peuvent être prises à partir des résultats d'un modèle, il est essentiel d'assurer une certaine transparence et d'expliquer les résultats aux utilisateurs finaux. Dans ce cas, des outils comme l'importance des variables ou les graphiques de type SHAP (SHapley Additive exPlanations) peuvent être utilisés pour améliorer l'interprétabilité.
- **Applications pratiques et optimisation continue :** Bien que le modèle Random Forest offre de bonnes performances, il est important de continuer à l'optimiser en ajustant ses hyperparamètres (comme le nombre d'arbres, la profondeur des arbres, et les échantillons minimums par feuille). Une optimisation continue est particulièrement utile dans les environnements où de nouvelles données sont régulièrement ajoutées et où les relations sous-jacentes peuvent évoluer.

- **Recommandations générales pour les applications réelles :**
  - **Exploration de modèles hybrides :** Une approche potentiellement intéressante consiste à utiliser une combinaison de modèles, où la régression linéaire pourrait être utilisée pour une première analyse rapide des données, suivie de l'application de modèles plus complexes comme Random Forest ou des réseaux neuronaux pour affiner les prédictions. Cette stratégie permet de tirer parti des avantages de plusieurs techniques tout en contournant les limites des modèles individuels.
  - **Préparation des données :** Une bonne préparation des données est cruciale pour garantir des modèles performants. Cela inclut la gestion des données manquantes, l'élimination des valeurs aberrantes et la normalisation des données si nécessaire. Les techniques de feature engineering peuvent également améliorer la capacité des modèles à capturer les relations complexes entre les variables.
  - **Analyse continue et mise à jour des modèles :** Dans des applications réelles, il est essentiel de surveiller régulièrement les performances du modèle et de le mettre à jour si nécessaire, notamment si les relations dans les données évoluent au fil du temps. Une mise à jour régulière des modèles garantit qu'ils restent pertinents et performants dans des contextes changeants.

Conclusion générale

Le projet a permis d'explorer en profondeur l'analyse des données musicales à travers des méthodologies de data mining. Dans le premier chapitre, la préparation des données et la sélection des caractéristiques ont été soigneusement abordées, mettant en lumière l'importance d'une bonne gestion des données pour garantir la qualité des analyses ultérieures.

Le deuxième chapitre a présenté les résultats des différentes modélisations, notamment l'utilisation de la régression linéaire et du modèle Random Forest. Les résultats ont révélé que la régression linéaire, bien que simple, n'était pas adaptée aux données en raison de sa faible capacité à expliquer la variance, comme en témoigne un Score R^2 de seulement 0.02. En revanche, le modèle Random Forest a montré des performances supérieures, bien qu'il ait également rencontré des limites.

Enfin, le troisième chapitre a permis une analyse critique des résultats, soulignant les implications pratiques des modèles utilisés et les défis rencontrés. Il a été conclu que, pour des données complexes, des modèles plus sophistiqués que la régression linéaire sont nécessaires pour capturer les relations sous-jacentes. Ce travail ouvre la voie à des recherches futures sur l'optimisation des modèles et l'exploration de nouvelles approches pour améliorer les recommandations musicales.

En somme, ce projet a non seulement mis en évidence les défis de l'analyse des données musicales, mais a également souligné l'importance d'une approche méthodologique rigoureuse pour obtenir des résultats significatifs et exploitables.