PROJET DATA CAMP

Groupe 9 :

- Marie WAHBA
- Angélique WAHBA
- Gaowen LI
- Estelle JOLAINE
- Walter TIMBA

Session 1 : Présentation de l'analyse du sujet et de la conception de notre solution

Sujet : Analyse des avis des utilisateurs sur Spotify grâce à leurs commentaires sur google play.

Contexte : Spotify aimerait avoir des informations et savoir ce qui est dit sur sa plateforme de streaming musical. Les utilisateurs laissent des avis sur leurs expériences, ce qui constitue une mine d'informations sur les fonctionnalités, les points positifs et négatifs de la plateforme.

Objectif principal : Utiliser des techniques d'analyse de sentiment pour identifier les perceptions des utilisateurs et suggérer des améliorations potentielles.

RACI :

Tâches	**Exécutant**
Collecte des données / scraping	Estelle JOLAINE
Prétraitement et nettoyage	Gaowen LI
Analyse de sentiment	Walter TIMBA

Tâches	Exécutant
Génération des résultats et propositions d'amélioration	Marie WAHBA Angélique WAHBA Gaowen LI Estelle JOLAINE Walter TIMBA
Visualisation des données	Angélique WAHBA
Validation des livrables	Marie WAHBA

Fonctionnalités attendues :

- Identifier les utilisateurs dont les avis ont le plus d'impact sur la communauté.
- Analyser les sentiments des utilisateurs de spotify à partir de mots clés.
- Remonter les avis les plus négatifs qui ressortent avec les mots les plus répétés pour avoir une idée de ce qu'il faut améliorer.
- Avoir un dashboard qui décrit le nombre de commentaires négatifs.

Données nécessaires:

Avis des utilisateurs sur Spotify.

Sources des données :

- ☐ Dataset Kaggle : «Spotify App Reviews»
- ☑ Web Scraping de la page Spotify sur Google Play Store.

Métadonnées :

Commentaires; Date d'avis; Version de l'application ; Nombre de likes; Étiquettes de sentiments(généré grâce au modèle de classification)

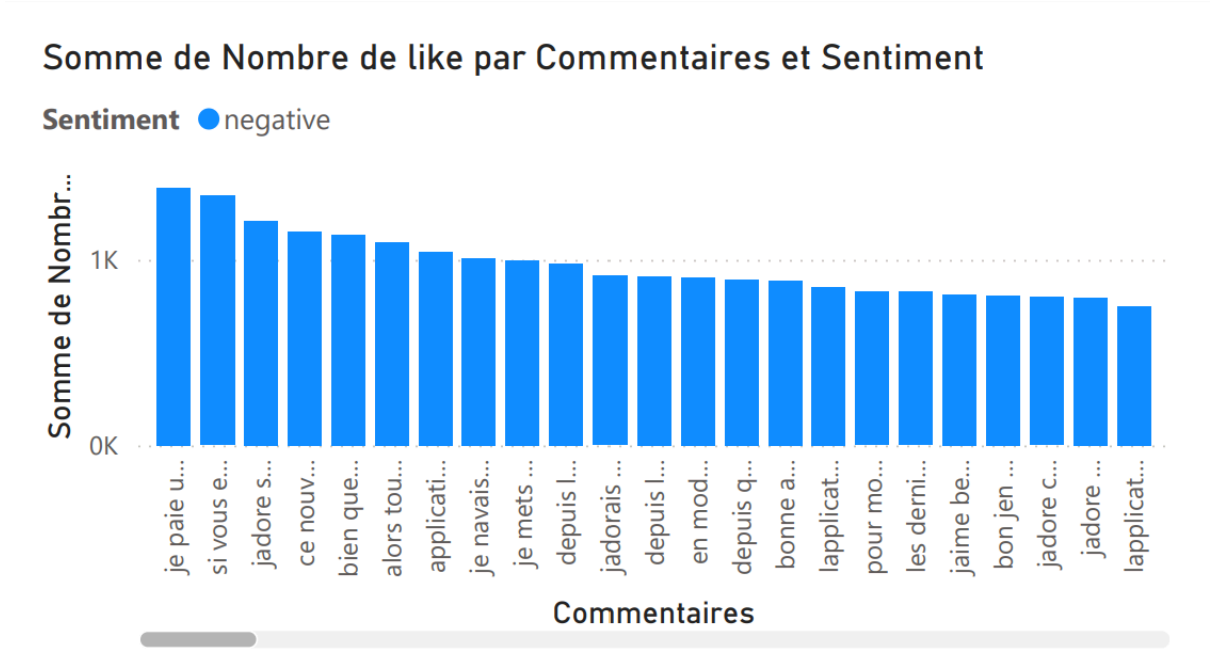
Etapes du projet :

- Collecte des données : Web Scraping / Kaggle
- Nettoyage et standardisation des données : Python; Jupyter Notebook; Pandas; Re
- Analyse des données : Python; Jupyter Notebook; TextBlob; Scikit-Learn
- Visualisation : Power BI; Tableau; Matplotlib
- Création d'une application web : Python, Streamlit, Transformers , Scipy, Matplotlib
- Hébergement dans le cloud : GitHub , StreamlitCloud, Rustc
- Rédaction : Google doc

Session 3 : Liste des fonctionnalités, les outils choisis et les algorithmes pour notre solution.

Liste des fonctionnalités :

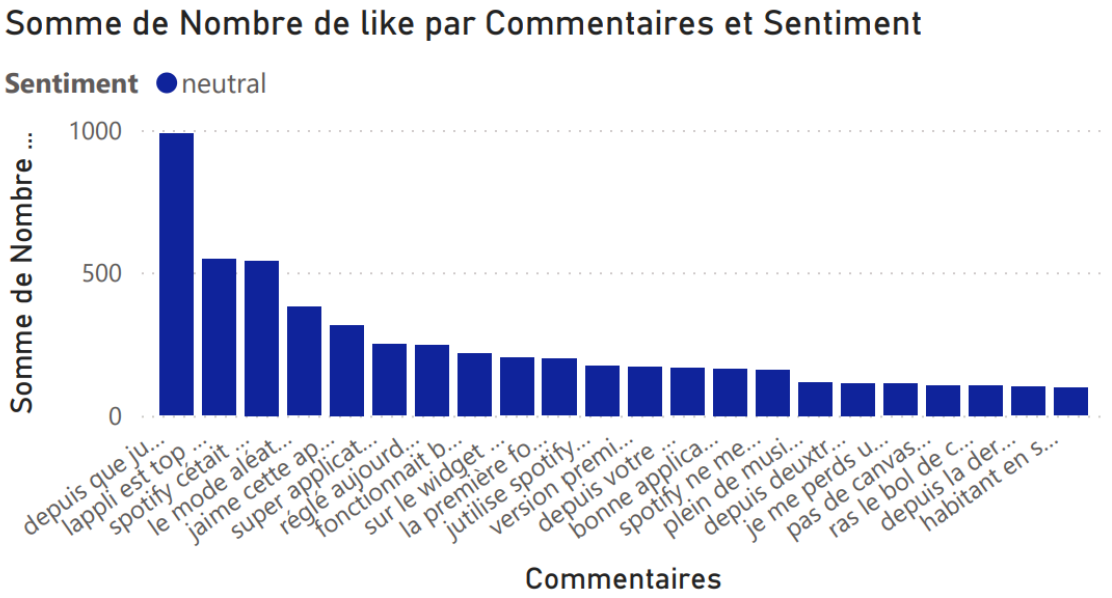
- Analyse la répartition des sentiments des commentaires :
  Identifier la proportion de commentaires positifs, neutres et négatifs pour évaluer globalement l'opinion des utilisateurs.
- Extraire les mots-clés pertinents :
  Identifier les termes ou expressions fréquemment mentionnés dans les commentaires pour détecter les thèmes récurrents ou préoccupations majeures.
- Classification des commentaires :
  Catégoriser automatiquement les commentaires pour faciliter leur analyse, notamment en identifiant les thèmes ou sous-catégories spécifiques.
- Visualisation des données :
  Présenter les résultats de manière claire et compréhensible dans un dashboard pour un public non technique.



Ce graphique montre les commentaires négatifs les plus "likés" par les utilisateurs. Cela indique que certains commentaires négatifs, bien que critiques, trouvent un écho auprès de nombreux autres utilisateurs. Par exemple :

Le commentaire *"je paie un abonnement à spotify depuis plusieurs années maintenant et jen étais pleinement satisfait jusquà il y a peu la gestion de la bibliothèque est devenue catastrophique on ne peut plus classer par titre tous les morceaux enregistrés se retrouvent dans un dossier like on met un temps fou à retrouver un titre les artistes des morceaux likés ne se retrouvent pas dans la catégorie artiste je pense sérieusement à résilier mon abonnement si pas de màj en conséquence"* semble être particulièrement significatif et a généré beaucoup d'engagement, ce qui peut indiquer un problème spécifique souvent partagé par les utilisateurs.

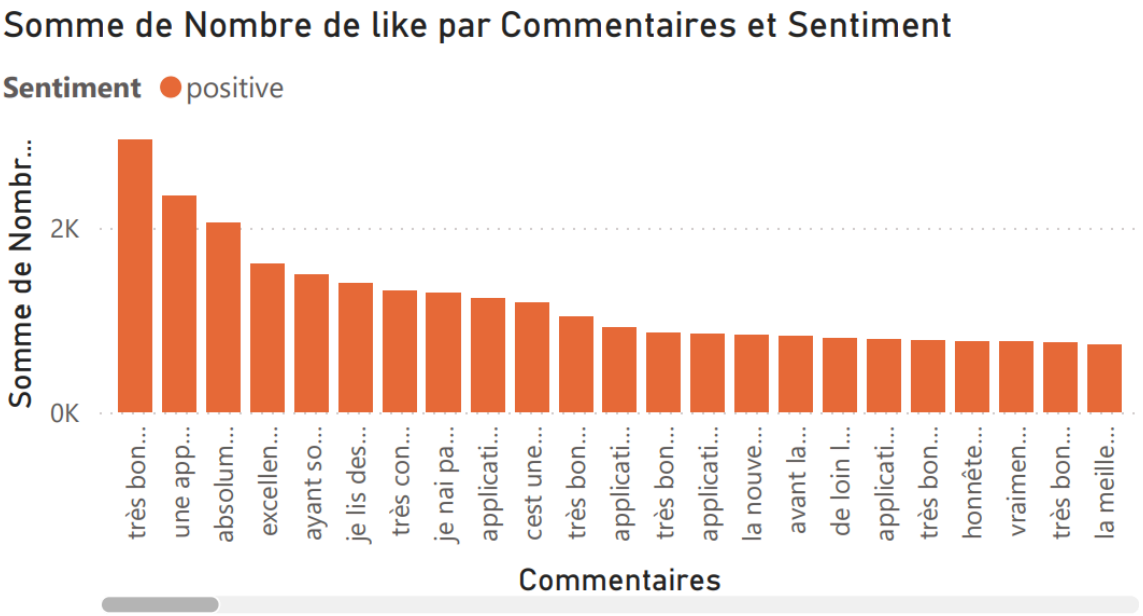
Les utilisateurs sont particulièrement réactifs à des points spécifiques négatifs, ce qui peut représenter des axes d'amélioration critiques pour Spotify.



Ce graphique met en évidence les commentaires neutres qui ont reçu un certain nombre de likes. Cela peut refléter des expériences partagées sans émotions extrêmes, mais qui restent importantes pour la communauté :

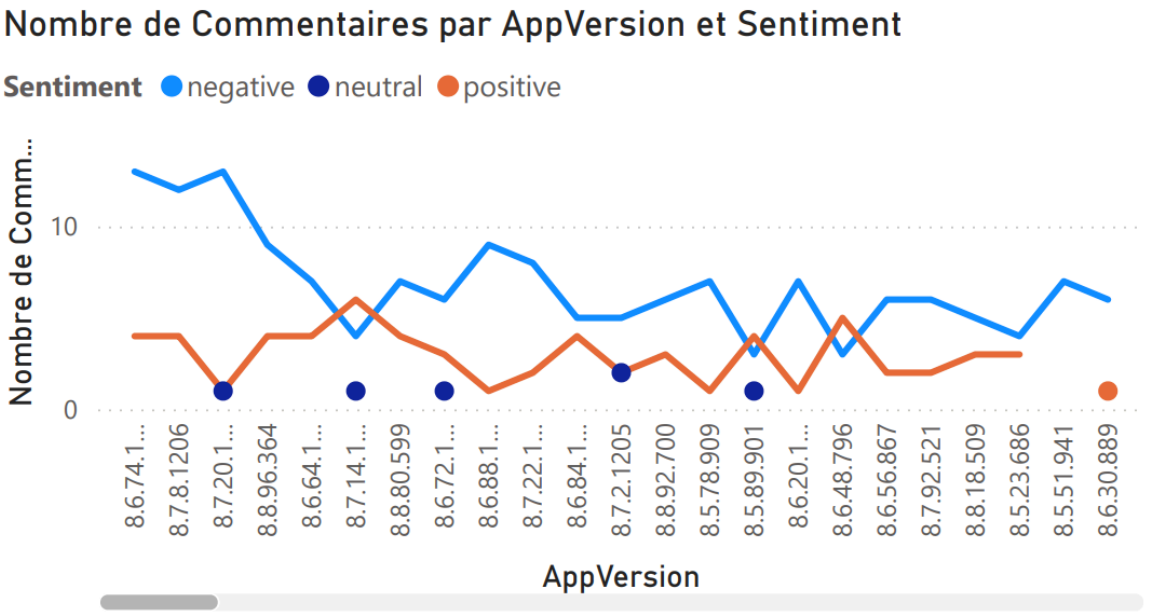
- Le commentaire *"depuis que jutilise spotify je nai pas rencontré de problème majeur mais depuis la dernière mise à jour impossible découter mes musiques en mode aléatoire pourtant le mode aléatoire est bien activé mais quand je veux recommencer un autre tour aléatoire la chanson se met seulement sur pause puis reprend sa lecture pareil quand je veux écouter un titre en particulier aucune possibilité de lancer une lecture aléatoire je nai pas dautre choix que découter les chansons dans lordre"* est le plus liké parmi les avis neutres, ce qui peut signaler des observations générales ou des suggestions d'amélioration passées inaperçues.

Bien que neutres, ces commentaires peuvent contenir des idées ou observations importantes qui méritent d'être étudiées.



Ici, les commentaires positifs les plus appréciés sont affichés. Les termes comme "très bon..." et "une app..." dominent, soulignant les aspects que les utilisateurs apprécient le plus chez Spotify, tels que la qualité du service ou des fonctionnalités spécifiques.

Cela démontre les forces perçues par les utilisateurs, qui pourraient être mises en avant dans la stratégie marketing ou consolidées dans le développement futur.



Ce graphique représente le nombre de commentaires classés par sentiment (positif, neutre, négatif) pour différentes versions de l'application. On observe :

- Un volume plus élevé de commentaires négatifs pour certaines versions spécifiques.
- Les versions avec des commentaires négatifs plus marqués (comme la version 8.6.671...) peuvent indiquer des mises à jour problématiques ou des régressions techniques.

Cette analyse montre un lien entre la version de l'application et les retours utilisateurs, ce qui peut aider à identifier les mises à jour nécessitant des ajustements.

- Fournir des recommandations d'amélioration par rapport aux versions antérieures de l'application : Basé sur les analyses précédentes, formuler des recommandations concrètes pour améliorer le produit ou service.

Outils utilisés :

- **Web scraping :**
  Google Play Scraper : un outil spécialement conçu pour extraire les avis depuis la page web Google Play.

- **Data cleaning :**
  Pandas : l'une des bibliothèques Python les plus utilisées pour manipuler et nettoyer les données.
  Re : Bibliothèque Python qui nous à permis grâce à la fonction clean text de retirer les caractères spéciaux et les espaces de trop dans les commentaires.

- **Modeling :**
  RoBERTa transformers : une amélioration de BERT qui est connu pour la classification textuelle et d'analyse des sentiments.

- **Visualisation :**
  Matplotlib et WordCloud : Utilisés pour représenter graphiquement les résultats de l'analyse.

PowerBI: Nous avons également utilisé PowarBI afin d'avoir un bon nombre de graphiques utiles pour notre analyse .

Les algorithmes :

Nous avons choisi RoBERTa pour les tâches suivantes :

- Classification de texte (par exemple, analyse des sentiments).
- Compréhension contextuelle (capacité à capturer les informations sémantiques dans les commentaires).
- Extraction de points clés (par exemple, identification des principaux problèmes ou fonctionnalités mentionnés dans les avis).

Pourquoi nous avons choisi RoBERTa **:**

- Performance exceptionnelle :
  RoBERTa peut détecter les nuances émotionnelles dans les commentaires, ce qui est crucial pour analyser les sentiments positifs, neutres et négatifs.
  Pour des ensembles de données similaires, RoBERTa dépasse souvent les modèles traditionnels de machine learning (comme la régression logistique).

- Adapté aux petits ensembles de données :
  Avec les modèles pré-entraînés de Hugging Face, RoBERTa peut être affiné sur de petits ensembles de données et obtenir rapidement de bonnes performances en classification.

- Capacité de compréhension contextuelle élevée :
  Pour les commentaires complexes, RoBERTa comprend le contexte et le sens global des phrases, plutôt que de se concentrer uniquement sur les mots individuels.

Défis rencontrés :

- Problème de scraping :

  Première tentative : Utilisation de BeautifulSoup (cf : TestScrapingSpotify.ipynb)

  Pour commencer, nous avons utilisé la bibliothèque **BeautifulSoup** pour effectuer le scraping des avis depuis la page de l'application Spotify sur le Google Play Store.

  - ➢ **Objectif** : Extraire les avis des utilisateurs directement depuis la page web de Spotify sur le Google Play Store.
  - ➢ **Méthodologie** :
    - Nous avons envoyé une requête HTTP avec la bibliothèque requests pour récupérer le contenu HTML de la page.
    - À l'aide de BeautifulSoup, nous avons analysé le contenu HTML pour identifier et extraire les sections contenant les avis.
    - Nous avons cherché les conteneurs et les classes spécifiques aux avis pour extraire des informations comme les notes, commentaires, dates, et likes.
  - ➢ **Résultat** :
    - **Problème** : La structure HTML de la page contenait des éléments générés dynamiquement via JavaScript, ce qui empêche leur récupération avec BeautifulSoup.
    - **Conclusion** : Nous avons réalisé que BeautifulSoup seul ne pouvait pas gérer le chargement dynamique des contenus générés par JavaScript.

  Deuxième tentative : Utilisation de SerpAPI (cf : Spotify.serpapi.py)

  Pour contourner les limitations rencontrées avec BeautifulSoup, nous avons essayé d'utiliser **SerpAPI**, une API spécialisée dans l'extraction de données de Google Play Store.

  - ➢ **Objectif** : Automatiser la récupération des avis via l'API SerpAPI.
  - ➢ **Méthodologie** :

- Nous avons généré une clé API en nous inscrivant sur SERPAPI.
- Nous avons récupéré l'ID de Spotify sur la page google playstore.
- Les avis obtenus étaient structurés sous forme de dictionnaires, que nous avons convertis en un DataFrame pour les analyser.

➢ **Résultat** :

- **Problème** : Malgré nos efforts, SerpAPI a retourné un faible nombre de résultats ou des données incomplètes, probablement en raison de restrictions ou d'un mauvais paramétrage de la requête.
- **Conclusion** : SerpAPI s'est révélé insuffisant pour couvrir nos besoins en matière de volume et de qualité des données.

Troisième tentative : Utilisation de google_play_scraper (cf: google_play_scraper.ipynb)

Enfin, nous avons opté pour **google_play_scraper**, une bibliothèque Python conçue pour extraire des données directement depuis Google Play Store.

➢ **Objectif** : Extraire les avis des utilisateurs via une solution dédiée et spécialisée.

➢ **Méthodologie** :

- Nous avons utilisé la fonction reviews pour récupérer un grand volume d'avis pertinents et récents.
- Les paramètres incluent le tri par pertinence, la langue française et une limite de 10 000 avis.
- Les données extraites ont été converties en un DataFrame pour une meilleure manipulation et analyse.

➢ **Résultat** :

- **Succès** : Cette solution a parfaitement fonctionné, nous permettant d'extraire un grand nombre d'avis avec toutes les informations nécessaires.
- **Conclusion** : google_play_scraper a répondu à nos attentes et nous a permis de poursuivre notre analyse.

Conclusion

BeautifulSoup nous a montré les limites des outils traditionnels pour des pages dynamiques.

SerpAPI nous a offert une alternative API mais a été limitée en termes de données fournies.

Google_play_scraper a été la solution idéale, simple et efficace, pour répondre à nos besoins.

- Choix du modèle : Initialement TextBlob a été testé pour effectuer l'analyse des données. Cependant, cette bibliothèque est insuffisante et moins précise pour répondre aux besoins du projet. C'est pourquoi nous avons choisi d'utiliser RoBERTa.

- Soucis de modeling : La compréhension et l'utilisation de RoBERTa ont présenté des difficultés, nécessitant l'installation de nombreuses bibliothèques pour surmonter ces obstacles.

Hébergement de l'application sur Streamlit Cloud

1. **Création du compte Streamlit Cloud et connexion à GitHub**
    - Nous avons créé un compte sur Streamlit Cloud, puis lié ce compte à GitHub en autorisant l'accès à nos dépôts via les paramètres de connexion de la plateforme.
2. **Mise en place du dépôt GitHub**
    - Un dépôt nommé `KeyceM2Moume/Groupe9_DataCamp` a été créé sur GitHub pour stocker le code source de notre application, incluant les fichiers nécessaires tels que `app.py`, `requirements.txt`, et d'autres fichiers associés.

3. **Difficulté rencontrée : configuration des dépendances**

    - Lors de la mise en place du fichier `requirements.txt`, des erreurs sont survenues lorsque certaines dépendances n'étaient pas correctement listées.
    - **Solution** : Nous avons soigneusement répertorié toutes les bibliothèques nécessaires, notamment `streamlit`, `transformers`, `torch`, `pandas`, etc., dans le fichier `requirements.txt`.
4. **Déploiement de l'application depuis GitHub**

    - **Difficulté rencontrée** : Des erreurs liées à des dépendances manquantes ou des versions incompatibles, comme l'installation de `tokenizers` nécessitant l'ajout de Rust, ont perturbé le déploiement.
    - **Solution** :
        - Mise à jour des bibliothèques dans le fichier `requirements.txt`.
        - Installation des outils additionnels, tels que Rust, pour résoudre les problèmes de compilation.
5. **Installation de Rust**

La commande suivante a été utilisée pour installer Rust via le terminal (Bash) :
Invoke-WebRequest -Uri https://sh.rustup.rs -OutFile rustup-init.exe; .\rustup-init.exe

    -
    - Nous avons suivi les instructions d'installation et ajouté Rust au PATH.
6. **Relance et finalisation**

    - Après avoir corrigé les problèmes, nous avons redémarré l'application, ce qui a permis de finaliser son déploiement sur Streamlit Cloud.