Université Lumière Lyon2

MASTER 2 SISE
Rapport

Analyse des Avis TripAdvisor sur les Restaurants Lyonnais

Étudiants :
Hugo COLLIN
Awa KARAMOKO
Alexis GABRYSCH
Falonne KPAMEGAN

Enseignant :
Ricco RAKOTOMALALA

19 janvier 2025

Table des matières

Introduction

Dans une société où le numérique occupe une place centrale, les avis en ligne sont devenus une source d'information essentielle pour les consommateurs. En particulier, le domaine de la restauration a vu émerger des plateformes comme TripAdvisor, qui permettent aux utilisateurs de découvrir des établissements, de partager leurs expériences et de guider les décisions des futurs clients. Ces avis, riches en contenu textuel, constituent une base précieuse pour analyser les perceptions des consommateurs, identifier des tendances, et explorer les aspects qualitatifs qui influencent la réputation des restaurants.

Lyon, connue mondialement comme la capitale de la gastronomie, offre un cadre idéal pour mener une telle étude. La diversité de ses restaurants, des bouchons traditionnels aux établissements étoilés, reflète une richesse culinaire unique. Cela en fait un terrain propice pour examiner comment les clients perçoivent ces établissements et comment ces perceptions se traduisent dans leurs évaluations en ligne.

Objectifs et enjeux

Le présent projet vise à exploiter les données issues de TripAdvisor pour analyser en profondeur les avis des utilisateurs sur une sélection de restaurants lyonnais. En combinant des techniques avancées de web scrapping, de modélisation de données, de traitement automatique du langage naturel (NLP) et d'intelligence artificielle (LLM), il s'agit de construire une solution complète, allant de l'acquisition des données à leur analyse et à leur visualisation interactive.

Les enjeux de ce projet sont multiples. Tout d'abord, structurer et organiser les données de manière à rendre leur exploitation efficace constitue une étape essentielle. Ensuite, l'analyse des avis permet de révéler des informations précieuses sur les sentiments des utilisateurs, les aspects les plus fréquemment commentés (comme le service, l'ambiance ou la qualité des plats) et les tendances globales ou spécifiques aux restaurants étudiés. Enfin, la conception d'une application interactive vise à rendre ces analyses accessibles et dynamiques, offrant ainsi un outil utile pour explorer et comparer les performances des différents établissements.

Organisation du rapport

Ce rapport présente l'ensemble du travail réalisé dans le cadre de ce projet. Après avoir exposé l'architecture et la méthodologie employées, les différentes étapes seront abordées en détail, depuis l'acquisition des données jusqu'à leur déploiement final sous forme d'une application interactive. Une discussion approfondie sur les résultats obtenus permettra de mettre en perspective les apports du projet, avant de conclure sur des pistes d'amélioration et des perspectives futures.

1 Architecture

Le code est structuré en deux grandes parties : **pages** qui contient les différentes pages de l'application et **src** qui regroupe les traitement sur la base de données (**db**), le traitement sur les textes (**nlp**) et l'algorithme de scraping (**searchengine**).

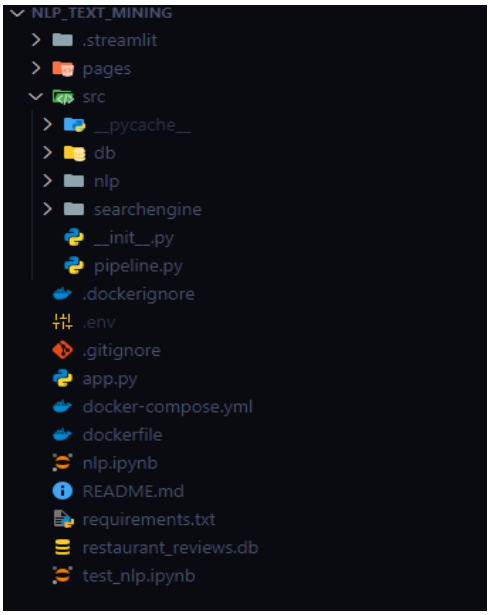

FIGURE 1 – Organisation du code du projet

2 Méthodologie

Dans cette section, nous décrivons la méthodologie adoptée pour collecter les données des restaurants à partir de la plateforme *TripAdvisor*. Cette méthodologie repose sur l'utilisation de techniques de *web scraping* en Python, en utilisant les bibliothèques `requests` et `BeautifulSoup`.

2.1 Collecte des Données

La collecte des données est le pilier central du projet de recommandation de restaurants. Elle permet d'acquérir des informations précises et actualisées sur les établissements, notamment les avis des utilisateurs, les notes attribuées, les coordonnées, les menus, et autres détails pertinents. En recueillant ces données, nous pouvons les croisées et ainsi en créer de nouvelles. Cependant, l'humain ne perçoit pas les données de la même façon qu'un ordinateur. Là où l'œil capte l'information, l'ordinateur lui reçoit du texte plus ou moins structuré et codifiée. C'est de ce postula que le scraping, méthode d'extraction automatique d'informations à partir de sites web en fait une méthode fastidieuse. En effet, nous avons rencontré un bon nombre de soucis et cela à plusieurs moment du projet notamment dû par l'adaptation des interfaces web à l'utilisateur.

Le processus de scraping a été structuré en trois classes principales :

— `SearchEngine` : Cette classe gère la navigation et l'extraction des URLs de chaque page de recherche.

— `RestaurantFinder` : Héritant de `SearchEngine`, elle permet de recueillir les Noms et URL de tous les restaurants sur Lyon de Tripadvisor. Elle se base sur l'identification des boites englobants les informations qui sont identiques à l'instar des données.

— `restaurant_info_extractor` : Héritant de `SearchEngine`, elle est responsable de l'extraction des données spécifiques à chaque restaurant, notamment les avis, les

détails et les images.

Dans certaines pages de restaurants, certaines informations essentielles, telles que les coordonnées ou les horaires, étaient absentes ou mal formatées. Cette absence de données entraîne des erreurs dans le processus de scraping et compromet la qualité des recommandations générées par l'application. Pour pallier ce problème, nous avons mis en place des vérifications conditionnelles avant l'extraction des données. Si une section spécifique est absente, le code attribue une valeur par défaut notamment `None` pour qu'on puisse plus facilement gérer les valeurs manquantes par la suite.

De plus, nous avons intégré des expressions régulières pour capturer les variations textuelles possibles, assurant ainsi une extraction plus robuste et flexible des données.

Les structures HTML des pages de TripAdvisor peuvent varier non seulement entre les pages de restaurants mais aussi entre les différentes sections d'une même page. Cette variation complique l'extraction systématique des données.

Pour gérer ces variations, nous avons adopté une approche modulaire dans le code de scraping. Chaque type d'information (notes, avis, détails) est extrait via des fonctions dédiées prenant en compte les différentes structures possibles. L'utilisation de sélecteurs CSS multiples et de méthodes de recherche alternatives (comme find_all avec différents critères) augmente la flexibilité et la résilience du scraper face aux changements de structure HTML. Ainsi, pour réaliser un seul scrapping des informations, on a l'utilisation de trois (03) en-têtes HTTP simulant trois navigateurs. Une première en version mobile pour obtenir les informations des localisation notations ainsi que les spécificités du restaurant, une autre sous `Mozilla Firefox` pour récupérer tous les avis et enfin axé sur `Chrome` pour récupérer des informations plus uniques comme le **Rang** ou les **horaires** du restaurant sur TripAdvisor. Cette methode couplée à l'utilisation de fonctions récursives à nombre d'itération prédefini en fait un outil robuste.

2.1.1 Processus de déroulement su scrapping

1. **Initialisation de la session de scraping :** Une session `requests` a été configurée pour inclure des en-têtes HTTP simulant un navigateur web classique (ex. `User-Agent`), afin d'assurer une navigation fluide et éviter les blocages liés à la détection de bots. Nous avons également intégré des cookies générés dynamiquement pour chaque requête.
2. **Gestion de la pagination :** Une fonction a été implémentée pour détecter et extraire l'URL de la page suivante à partir des balises HTML associées à la navigation (`aria-label="Page suivante"`). Lorsque la dernière page est atteinte, l'extraction s'arrête automatiquement. Cette gestion est la même pour le scraping des URL des restaurants comme pour celui des avis utilisateurs sur plusieurs pages.
3. **Extraction des informations principales :** Sur chaque page de restaurant, des informations clés telles que le nom, l'adresse, les coordonnées, les notes, les types de cuisines, les gammes de prix, et les régimes alimentaires spéciaux ont été extraites. Une attention particulière a été portée aux étoiles Michelin, identifiées via les images associées.
4. **Extraction des avis :** Les avis des utilisateurs ont également été collectés, comprenant les titres des avis, leur contenu, la date de publication, la note attribuée, ainsi que des détails sur le type de visite (ex. en couple, entre amis).

5. **Gestion des erreurs et des blocages :** Des mécanismes ont été intégrés pour gérer les erreurs courantes (ex. codes HTTP 403, 429) via des temporisations, changement de sessions et une limitation du nombre de requêtes successives puisque l'hote du scraper est soumis à de potentielles restrictions d'IP si nous requetons trop souvent .

Les données collectées ont été stockées dans une base de données pour une analyse ultérieure. À la fin du processus de scraping, un total de **2953 restaurants** ont été collecté, dont **32** incluant leurs informations détaillées et plus de **20000 avis**. Ces données serviront de base pour les analyses dans les sections prochaines et sont susceptibles d'être augmentées.

2.2 Base de données

2.2.1 Modélisation de la base de données

La base de données du projet repose sur un modèle en étoile. Ce modèle est adapté aux applications analytiques, car il simplifie les requêtes complexes et optimise les performances. Elle est constituée d'une table de faits (`Reviews`) et de plusieurs tables de dimensions (`users`, `restaurants`).

La table de faits centralise les avis laissés par les utilisateurs sur les restaurants. Nous allons pouvoir analyser **les notes moyennes**, **les meilleurs contributeurs**, **les sentiments des avis** et **les tendances de visites** par restaurants.

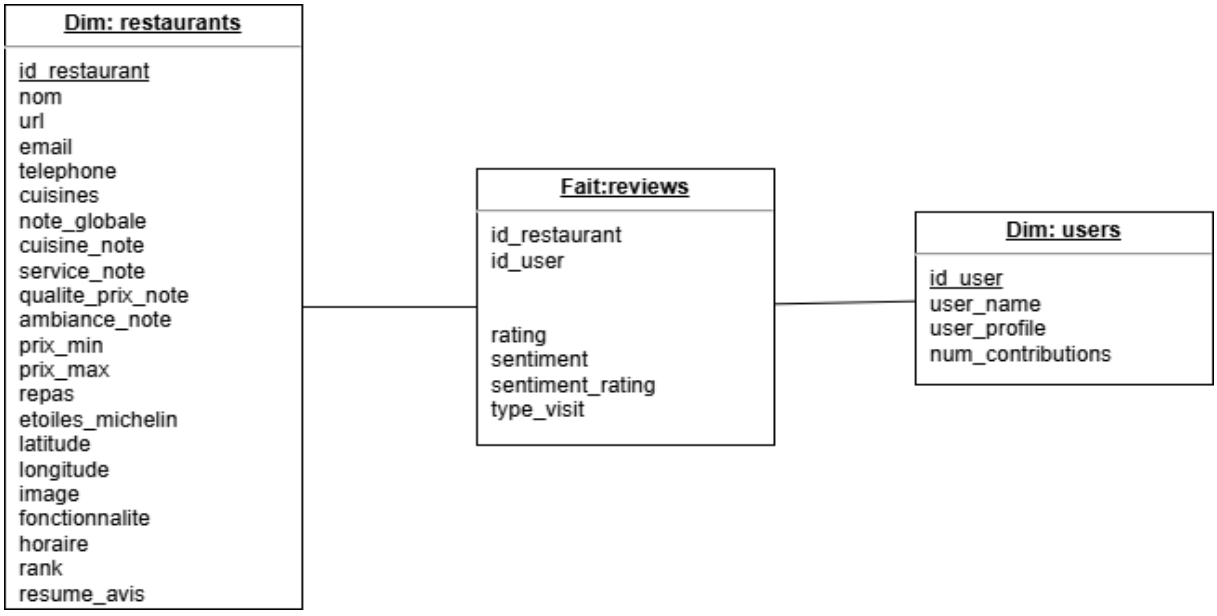

FIGURE 2 – Modèle en étoile de l'entrepôt de données

2.3 Structure des tables

2.3.1 Table des utilisateurs (`dim_users`)

La table `dim_users` contient les informations sur les utilisateurs qui laissent des avis sur les restaurants. Elle est définie comme suit :

Listing 1 – Création de la table `dim_users`

```
CREATE TABLE dim_users (
    id_user INTEGER PRIMARY KEY AUTOINCREMENT,
    user_name TEXT NOT NULL,
    user_profile TEXT,
    num_contributions INTEGER
);
```

Les colonnes principales incluent :

— `id_user` : Identifiant unique de l'utilisateur.
— `user_name` : Nom de l'utilisateur.
— `user_profile` : Profil de l'utilisateur (par exemple, lien vers son profil TripAdvisor).
— `num_contributions` : Nombre total de contributions de l'utilisateur.

2.3.2 Table des restaurants (`dim_restaurants`)

La table `dim_restaurants` contient les informations relatives aux restaurants. Voici son schéma SQL :

Listing 2 – Création de la table `dim_restaurants`

```
CREATE TABLE dim_restaurants (
    id_restaurant INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL,
    adresse TEXT,
    url_link TEXT,
    email TEXT,
    telephone TEXT,
    cuisines TEXT,
    note_globale FLOAT,
    cuisine_note FLOAT,
    service_note FLOAT,
    qualite_prix_note FLOAT,
    ambiance_note FLOAT,
    prix_min FLOAT,
    prix_max FLOAT,
    etoiles_michelin INTEGER,
    repas TEXT,
    latitude FLOAT,
    longitude FLOAT,
    fonctionnalite TEXT,
    horaires TEXT,
    rank TEXT
    image TEXT,
    resume_avis TEXT
);
```

Les colonnes incluent :

— `id_restaurant` : Identifiant unique du restaurant.
— `nom`, `adresse`, `email`, `telephone` : Informations générales sur le restaurant.

— `cuisines` : Type(s) de cuisine proposée(s) (par exemple, française, italienne).
— `note_globale`, `cuisine_note`, `service_note`, etc. : Notes attribuées par les utilisateurs sur différents critères.
— `prix_min` et `prix_max` : Fourchette de prix des repas.
— `etoiles_michelin` : Nombre d'étoiles Michelin.
— `resume_avis` : Résumé automatique des avis (ajouté dynamiquement après traitement NLP).

2.3.3 Table des avis (`fact_reviews`)

La table `fact_reviews` centralise les avis et relie les dimensions des utilisateurs et des restaurants. Elle est définie comme suit :

Listing 3 – Création de la table `fact_reviews`

```
CREATE TABLE fact_reviews (
    id_review INTEGER PRIMARY KEY AUTOINCREMENT,
    id_restaurant INTEGER NOT NULL,
    id_user INTEGER NOT NULL,
    date_review DATE,
    title_review TEXT,
    review_text TEXT,
    rating FLOAT,
    type_visit TEXT,
    review_cleaned TEXT,
    sentiment INTEGER,
    sentiment_rating TEXT,
    FOREIGN KEY (id_restaurant) REFERENCES dim_restaurants(
       id_restaurant),
    FOREIGN KEY (id_user) REFERENCES dim_users(id_user)
);
```

Les colonnes incluent :
— `id_review` : Identifiant unique de l'avis.
— `id_restaurant`, `id_user` : Clés étrangères vers les tables de dimensions.
— `date_review`, `title_review`, `review_text` : Informations sur l'avis.
— `rating` : Note globale attribuée par l'utilisateur.
— `type_visit` : Type de visite (famille, amis, affaires, etc.).
— `review_cleaned` : Texte de l'avis après nettoyage.
— `sentiment`, `sentiment_rating` : Analyse du sentiment NLP (score numérique et étiquette correspondante, par ex. positif, neutre, négatif).

2.4 Choix du SGBD : SQLite

Pour ce projet, nous avons opté pour l'utilisation de SQLite comme système de gestion de base de données (SGBD). Ce choix a été motivé par plusieurs facteurs techniques et pratiques qui répondent parfaitement aux besoins du projet.

2.4.1 Avantages de SQLite pour ce projet

- **Légèreté et simplicité** : SQLite est un SGBD léger qui ne nécessite pas de serveur dédié. Cela simplifie considérablement l'installation et l'exécution, particulièrement dans le cadre de projets déployés via Docker, où la gestion des dépendances doit être optimisée.
- **Structure autonome** : Contrairement aux bases de données client-serveur comme MySQL ou PostgreSQL, SQLite fonctionne sur un simple fichier. Cela facilite la manipulation, le partage et la sauvegarde de la base de données.
- **Performance adaptée** : Pour un volume modéré de données, comme c'est le cas ici avec une trentaine de restaurants et leurs avis associés, SQLite offre des performances suffisantes. Il est capable de gérer efficacement les requêtes SQL complexes, y compris les jointures entre plusieurs tables.
- **Facilité d'intégration** : SQLite s'intègre parfaitement avec Python grâce à la bibliothèque standard `sqlite3`. Cela simplifie la connexion entre la base de données et l'application développée en Streamlit, tout en minimisant les dépendances supplémentaires.
- **Portabilité** : L'absence de dépendances externes rend SQLite très portable. Le fichier de base de données peut être facilement transféré entre différents systèmes ou inclus dans l'image Docker du projet.
- **Respect des spécifications du projet** : SQLite répond aux exigences de ce projet, qui implique la récupération, le stockage structuré et l'analyse de données. Sa compatibilité avec les modèles relationnels, notamment le modèle en étoile utilisé ici, garantit une gestion robuste des données.

2.5 Modèle ORM : Gestion des données avec SQLAlchemy

Pour faciliter les interactions entre l'application Python et la base de données SQLite, nous avons adopté un modèle ORM (Object-Relational Mapping) à l'aide de la bibliothèque `SQLAlchemy`. Cette approche permet de manipuler les données de manière intuitive, en travaillant avec des objets Python au lieu de manipuler directement des requêtes SQL.

2.5.1 Représentation des tables sous forme de classes

Dans le modèle ORM, chaque table de la base de données est représentée par une classe Python. Les colonnes des tables sont directement mappées aux attributs de ces classes. Par exemple, les tables `dim_users`, `dim_restaurants`, `fact_reviews` sont représentées par les classes `User`, `Restaurant` et `Review` respectivement.

Ces classes permettent de représenter les données sous une forme orientée objet, rendant leur manipulation plus claire et intuitive.

L'utilisation d'un modèle ORM présente plusieurs avantages dans le cadre de ce projet :

- **Lisibilité et simplicité du code** : Les données peuvent être manipulées directement comme des objets Python (par exemple, un avis est une instance de la classe `Review`), ce qui simplifie grandement le code par rapport à l'écriture de requêtes SQL brutes.
- **Gestion des relations entre les tables** : Les relations entre les tables (par exemple, un utilisateur peut laisser plusieurs avis, chaque avis est lié à un restaurant) sont représentées naturellement dans le modèle objet. Cela permet de

naviguer facilement entre les entités sans écrire de requêtes complexes.
— **Abstraction de la base de données** : L'ORM masque les détails spécifiques de l'implémentation SQL. Il offre une couche d'abstraction qui facilite la migration ou la modification du SGBD si nécessaire.
— **Requêtes dynamiques** : Il est possible d'effectuer des requêtes complexes (filtres, tris, agrégations) directement via les méthodes de l'ORM, tout en bénéficiant d'une syntaxe claire et expressive.

3 Analyse NLP

Dans la partie consacrée au traitement automatique du langage naturel (NLP), nous distinguons deux grandes étapes principales : le prétraitement des données et leur analyse. Pour répondre à ces besoins, nous avons développé deux classes :

— **NLPPretraitement** : dédiée aux opérations de prétraitement, telles que le nettoyage des avis.
— **NLPAnalysis** : conçue pour l'analyse des données textuelles, incluant des tâches comme l'analyse des sentiments et la génération de résumés automatiques des avis.

3.1 Présentation de la Classe `Pretraitement`

La classe `Pretraitement` est conçue pour effectuer une série d'opérations sur les avis textuels collectés, allant du nettoyage initial à l'analyse des sentiments. Elle intègre plusieurs méthodes spécialisées qui, en synergie, préparent les données brutes.

3.1.1 Méthode `nettoyer_avis`

La méthode `nettoyer_avis` a pour objectif de nettoyer les avis utilisateurs en supprimant les éléments indésirables et en normalisant le texte pour une analyse ultérieure.

3.1.2 Processus

— **Conversion en minuscules** : L'ensemble de l'avis est converti en minuscules pour assurer une uniformité dans l'analyse.
— **Suppression des chiffres** : Les chiffres sont retirés du texte à l'aide d'expressions régulières (`re.sub`), car ils peuvent ne pas apporter de valeur significative à l'analyse de sentiment.
— **Tokenisation** : Les avis sont découpés en tokens individuels (mots) en utilisant la méthode `word_tokenize` de la bibliothèque NLTK.
— **Lemmatisation** : Chaque mot est réduit à sa forme de base à l'aide d'un lemmatiseur, ce qui aide à standardiser les mots et à réduire la complexité du vocabulaire.
— **Filtrage des mots vides** : Les mots courants et non significatifs (*stop words*) sont supprimés pour se concentrer sur les termes pertinents.
— **Longueur des mots** : Seuls les mots de plus de deux caractères sont conservés, éliminant ainsi les abréviations et les termes non significatifs.

Enfin, les tokens nettoyés sont recombinés en une chaîne de caractères propre.

3.2 Méthode `preprocess`

3.2.1 Objectif

La méthode `preprocess` effectue un prétraitement similaire à `nettoyer_avis`, mais est conçue pour offrir une flexibilité supplémentaire dans le pipeline de traitement des données.

3.2.2 Processus

— **Conversion en minuscules** : Uniformisation des caractères.
— **Tokenisation** : Division du texte en mots individuels.
— **Filtrage des mots vides et des mots courts** : Élimination des termes non pertinents.
— **Lemmatisation** : Réduction des mots à leur forme de base.

Cette méthode renvoie une chaîne de caractères prétraitée, prête pour d'autres étapes d'analyse.

3.2.3 Méthode `preprocess_reviews`

3.2.4 Objectif

La méthode `preprocess_reviews` applique le prétraitement à un ensemble de données d'avis stocké dans un DataFrame pandas, automatisant ainsi le nettoyage pour un grand volume de données.

3.2.5 Processus

— **Suppression des valeurs manquantes** : Les avis nuls ou manquants sont retirés pour éviter les erreurs lors de l'analyse.
— **Application du prétraitement** : La méthode `preprocess` est appliquée à chaque avis pour le nettoyer et le normaliser.
— **Création d'une nouvelle colonne** : Les avis nettoyés sont stockés dans une nouvelle colonne `review_cleaned` du DataFrame.

Cette méthodologie permet de préparer efficacement les données pour les étapes ultérieures d'analyse de sentiment ou de modélisation.

3.2.6 Méthode `sentiment_analysis`

3.2.7 Objectif

La méthode `sentiment_analysis` analyse le sentiment des avis prétraités en déterminant s'ils sont positifs ou négatifs.

3.2.8 Processus

— **Calcul de la polarité** : Utilisation de la bibliothèque `TextBlob` pour évaluer la polarité de chaque avis nettoyé. La polarité est une valeur numérique indiquant la positivité ou la négativité du texte.
— **Classification des sentiments** : Les avis sont classés en deux catégories principales :

— *Positive* : Si la polarité est supérieure ou égale à zéro.
— *Negative* : Si la polarité est inférieure à zéro.
— **Agrégation des résultats** : Affichage de la répartition des sentiments à l'aide de la méthode `value_counts`, fournissant une vue d'ensemble des avis.

3.2.9 Vectorisation des Textes

La méthode `vectorisation` transforme les avis nettoyés en représentations numériques, nécessaires pour les modèles d'analyse NLP :
— **Bag-of-Words (BoW)** : Compte la fréquence des mots dans les textes.
— **TF-IDF (Term Frequency - Inverse Document Frequency)** : Pondère les mots en fonction de leur importance relative dans l'ensemble des documents.
Les paramètres permettent de limiter le nombre de caractéristiques avec `max_features`.

3.3 Analyse des données

3.3.1 Préprocessing des Avis

3.4 Méthode `summarize_reviews`

3.4.1 Objectif

La méthode `summarize_reviews` a pour objectif de condenser un grand nombre d'avis utilisateurs en un résumé clair et informatif. Ce résumé permet d'obtenir une vue d'ensemble rapide des opinions et des sentiments exprimés par les clients, facilitant ainsi la prise de décision et l'amélioration des services proposés par les restaurants.

3.4.2 Description du Processus

La méthode `summarize_reviews` suit plusieurs étapes clés pour générer un résumé efficace des avis utilisateurs :

1. **Initialisation du Modèle de Résumé**
   La méthode utilise la bibliothèque `transformers` de Hugging Face pour initialiser un pipeline de résumé basé sur le modèle `facebook/bart-large-cnn`. Ce modèle est pré-entraîné pour effectuer des tâches de génération de texte, y compris la synthèse de documents longs en résumés concis.
2. **Agrégation des Avis**
   Tous les avis contenus dans le DataFrame `df` sont concaténés en une seule chaîne de caractères. Cette agrégation permet de traiter l'ensemble des avis de manière unifiée lors de la génération du résumé.
3. **Découpage en Chunks**
   Étant donné que les modèles de résumé ont une limite de taille de texte (dans ce cas, 1024 caractères), les avis agrégés sont divisés en plusieurs segments appelés "chunks". Chaque chunk a une longueur maximale de 1024 caractères pour assurer une traitement efficace par le modèle de résumé.
4. **Sélection Aléatoire des Chunks**
   Pour gérer un volume potentiellement très important de chunks, un échantillon aléatoire de 20 chunks est sélectionné. Cette sélection aléatoire permet de diversifier

les informations prises en compte dans le résumé final tout en maintenant une charge de traitement raisonnable.

5. **Génération des Résumés pour Chaque Chunk**
   Chaque chunk sélectionné est passé au modèle de résumé, qui génère un résumé de 40 à 60 mots pour chaque segment. Cette étape produit des résumés intermédiaires qui capturent les points essentiels de chaque portion d'avis.
6. **Agrégation des Résumés Intermédiaires**
   Les résumés intermédiaires sont ensuite combinés en une seule chaîne de caractères, formant ainsi le résumé final des avis. Cette étape rassemble les informations clés extraites de chaque chunk, offrant une vue d'ensemble synthétique des avis utilisateurs.

Cette approche permet d'obtenir un résumé représentatif des avis tout en optimisant les ressources de calcul nécessaires à la génération du résumé.

4 Pipeline

4.1 Le Rôle Essentiel de la Classe `Transistor`

Dans le cadre de notre pipeline, la classe `Transistor` joue un rôle central et fondamental. Elle agit comme une interface qui relie efficacement les différents modules, fichiers, et la base de données, tout en assurant une gestion cohérente et simplifiée des ressources.

La `Transistor` encapsule des fonctionnalités telles que la gestion des sessions pour interagir avec la base de données. Au lieu d'initialiser des sessions SQLAlchemy individuellement dans chaque fichier ou classe, la `Transistor` centralise cette responsabilité. Cette approche réduit non seulement les risques d'erreurs liés à la gestion des sessions, mais aussi facilite leur réutilisation et leur fermeture appropriée.

Cette session est partagée de manière transparente entre les composants, ce qui permet des appels cohérents, comme `get_all_restaurants()` ou `insert_review()`, sans avoir à réinitialiser les sessions à chaque interaction.

Un autre point fort est l'abstraction des appels directs à la base de données. Par exemple, les méthodes comme `insert_user()` ou `update_restaurant()` sont redirigées vers des fonctions dédiées dans des modules externes (`update_db` ou `functions_db`), tout en maintenant la gestion des sessions dans la classe `Transistor`.

4.2 La Classe `Pipeline`

La classe **Pipeline** constitue une extension de la classe **Transistor**, avec pour objectif principal de coordonner et d'automatiser les différentes étapes du traitement des données. Tandis que **Transistor** établit les fondations nécessaires pour la gestion des sessions et des connexions entre les différents modules, **Pipeline** encapsule des processus complexes en intégrant ces modules dans un flux cohérent.

De plus, la `Pipeline` sert de *hub* pour orchestrer les connexions entre des modules variés tels que la recherche de données (`SearchEngine`), le traitement linguistique (`NLPPretraitement`), et la gestion des données (`process_restaurant_data`). Par exemple, lors de l'ajout d'un restaurant, la classe initie plusieurs composants :

Par exemple, la méthode `add_new_restaurant()` illustre cette orchestration en combinant :

— Le scraping des données d'un restaurant depuis son URL.
— La transformation des avis bruts en textes nettoyés.
— L'analyse des avis et la génération d'un résumé concis grâce à une API externe.
— L'insertion des résultats traités dans la base de données.

En simplifiant l'interface utilisateur et en automatisant les enchaînements d'étapes, **Pipeline** garantit une gestion efficace et fluide des flux de données, tout en exploitant pleinement les fonctionnalités offertes par **Transistor**.

5 Développement de l'Application Streamlit

L'application **SISE Ô Resto** a été conçue pour offrir une expérience culinaire optimale à Lyon. Elle permet à ses utilisateurs de découvrir facilement les meilleurs restaurants de la ville en fonction de leurs préférences et de leur localisation. Son interface simple et intuitive permet de rechercher, comparer et localiser des restaurants de manière fluide, facilitant ainsi la prise de décision avant chaque repas.

Les utilisateurs peuvent explorer une large sélection de restaurants lyonnais en **filtrant les résultats par critères** tels que le type de cuisine, la fourchette de prix ou encore les avis des clients. Cette fonctionnalité permet de personnaliser l'expérience en fonction des besoins et des goûts de chaque utilisateur. Pour chaque restaurant, l'application propose une fiche détaillée contenant des informations essentielles comme les horaires d'ouverture, le menu, les avis des utilisateurs, et les coordonnées du lieu, mais aussi une deuxième section consacrée à une analyse approfondie des avis laissés par les clients des restaurants sur TripAdvisor, comprennant des informations détaillés sur chaque avis et des graphiques d'analyses. Ces informations aident à prendre une décision éclairée avant de se rendre sur place.

Une autre fonctionnalité clé de l'application permet de **comparer différents restaurants entre eux**. En quelques clics, les utilisateurs peuvent comparer des restaurants selon divers critères (prix, type de cuisine, avis des clients), ce qui les aide à affiner leur choix en fonction de leurs priorités du moment.

L'application va au-delà des simples informations en offrant des **statistiques détaillées sur chaque restaurant**. Ces statistiques incluent des données sur la satisfaction des clients, la popularité de l'établissement et des comparaisons entre différents restaurants. Ces informations permettent aux utilisateurs de mieux comprendre l'offre culinaire et de faire un choix encore plus éclairé.

En termes de géolocalisation, l'application inclut une **carte interactive permettant aux utilisateurs de localiser facilement les restaurants à proximité**. Lors de la première utilisation, il leur suffit de renseigner leur adresse pour activer cette fonctionnalité. L'application propose ensuite le meilleur itinéraire en un clic pour rejoindre le restaurant sélectionné, rendant l'expérience encore plus pratique. En se rendant sur une fiche complète du restaurant nous obtenons même le temps de trajet pour chaque type d'itinéraire (mode doux, transports en communs et voiture). Pour cela un scraping sur le site des TCL a été mit en place. Cela n'a pas été une tache facile car nous voulions des données fraîches et en temps réel, donc nous ne pouvions pas prendre de dataset mit à disposition en ligne, et le site TCL ne propose pas d'API publique. Nous avons donc du créer un code se fessant passer pour le site TCL afin de récolter les résultats de la re-

quête que le site effectuait pour obtenir les temps de parcours (car les temps de parcours n'étaient pas directement récupérables depuis le code HTML du site). L'autre avantage de renseigner son adresse dans l'application est de pouvoir filtrer les restaurant selon un rayon autour du domicile afin de n'obtenir seulement les restaurants près de chez soi.

De plus en dessous de la carte, un graphique du clustering des restaurants en 3D selon 3 composantes principales est disponible. Enfin une fonctionnalité expérimentale a été ajoutée permettant aux utilisateurs de l'application de pouvoir créer ses propres graphiques selon plusieurs types de graphiques sur les données de la table Restaurants.

L'architecture de l'application se compose de trois pages principales.

— La première, la page d'**Accueil**, présente un aperçu des fonctionnalités et permet de configurer la localisation actuelle de l'utilisateur. **Il est important de renseigner sa localisation actuelle pour profiter pleinement des fonctionnalités de l'application**.

— La page **Explorer** offre une vue détaillée des restaurants de Lyon, avec des options de filtrage avancées pour affiner la recherche. Elle propose aussi un comparateur pour comparer les restaurants et aussi un espace pour discuter avec le chatbot Mistral "*tuné*" pour améliorer l'expérience utilisateur dans le cadre de la restauration lyonnaise. Il est également possible de générer et de visualiser des graphes personnalisés, adaptés aux différents angles d'analyse choisis par l'utilisateur, offrant ainsi une flexibilité dans l'interprétation des données.

— Enfin, la page **Admin** est dédiée aux gestionnaires du système et leur permet de gérer les restaurants inscrits, de mettre à jour les informations et d'effectuer des requêtes SQL simples. Cette page a eu pour but de nous aider grandement pour le développement de l'application, le débogage de celle-ci et le test.

Il est important de souligner le fait l'interface a été pensée dans le but d'obtenir une naviguation simple et intuitive pour chacun des utilisateurs. De plus nous avons consacré beaucoup de temps à tester l'application afin d'être sûr qu'il n'y a pas de bug ou d'effets inattendus/indésirables, dans le but de fournir une application de qualité.

FIGURE 3 – Interface streamlit

5.1 Intégration de Mistral AI

Le modèle Mistral joue un rôle essentiel dans la gestion des interactions entre l'application et les utilisateurs. Il est chargé de comprendre les préférences exprimées par ces derniers et de leur fournir des suggestions pertinentes basées sur les données disponibles. Les principales fonctionnalités offertes par Mistral incluent :

- **Analyse des préférences utilisateur :** Le modèle identifie les besoins exprimés par l'utilisateur, tels que le type de cuisine recherché, le budget, la localisation, ou encore les exigences particulières comme les options végétariennes ou l'accessibilité.
- **Recommandations personnalisées :** En s'appuyant sur la base de données de l'application, Mistral propose des restaurants correspondant aux critères définis par l'utilisateur. Les recommandations incluent des informations détaillées, comme le nom du restaurant, son adresse, les avis clients, les heures d'ouverture, et d'autres données pertinentes. Pour cela la base de données est transformé en **chuncks** associé à des *embeddings* puis enregistrés dans un table spécialement faite pour l'IA. Cela lui permet de construire ses propres connaissances de manière optimisée afin de comprendre et de pouvoir répondre au mieux aux questions de l'utilisateur.
- **Interaction naturelle et efficace :** Grâce à sa capacité de traitement du langage naturel, Mistral permet une interaction fluide avec les utilisateurs, posant des questions supplémentaires si nécessaire pour affiner les suggestions et répondant de manière claire et concise. De plus il apprends de la discussion en cours et est capable de continuer la discussion sur le thème de recherche choisi par l'utilisateur.

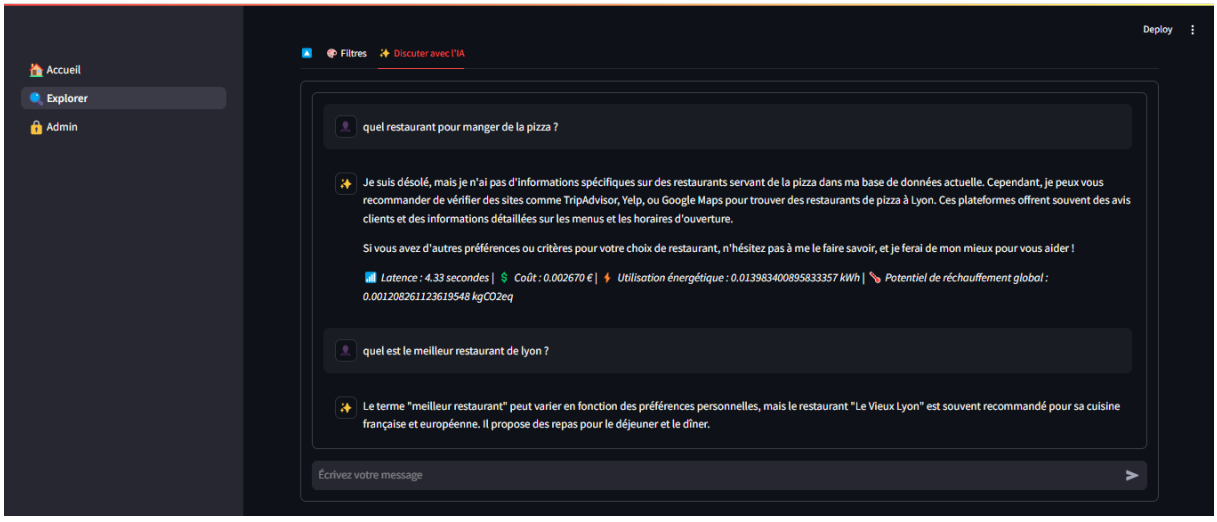

FIGURE 4 – Intégration de Mistral

5.1.1 Fonctionnement technique

L'implémentation du modèle Mistral repose sur une architecture bien structurée intégrée à l'application. Les principales étapes de son fonctionnement sont décrites ci-dessous :

- **Initialisation et gestion des messages :** Les messages des utilisateurs sont capturés et ajoutés à l'historique des conversations, stocké dans la session de l'application. Si l'utilisateur utilise le système pour la première fois, la base de données contenant des informations sur les restaurants lyonnais (`bdd_chunks`) est initialisée pour permettre des recherches rapides et pertinentes.

— **Chargement et configuration du modèle Mistral :** Le modèle est configuré avec un *prompt* spécifique qui définit son rôle et ses fonctionnalités. Il est exécuté avec des paramètres optimisés, comme une faible température (`0.3`) pour favoriser des réponses cohérentes, et une limite de `3000 tokens` pour traiter des échanges détaillés.
— **Affichage des résultats :** Les réponses générées par Mistral sont affichées dans une interface utilisateur conviviale. Ces réponses sont accompagnées de métriques de performance, comme la latence, le coût de la requête (ici nous ne payons rien puisque la clé nous as été donné gratuitement), la consommation énergétique, et l'impact environnemental (*Potentiel de Réchauffement Global*), pour suivre l'efficacité et la durabilité des interactions.

5.2 Dockerisation

Afin de garantir une portabilité optimale et une exécution homogène de l'application **SISE Ô Resto** sur différents environnements, nous avons utilisé Docker pour son déploiement. Docker permet de contenir l'application et toutes ses dépendances au sein d'une image légère et autonome, assurant ainsi une configuration cohérente entre les machines des développeurs et les environnements de production.

Les fichiers `docker-compose.yml` et `dockerfile` définissent les services nécessaires pour exécuter l'application. Un conteneur principal est configuré pour héberger l'application Streamlit, exposant le port `8501` pour l'accès utilisateur. Les variables d'environnement nécessaires à l'exécution de l'application, telles que la langue et les paramètres de Streamlit, sont également spécifiées pour garantir une expérience utilisateur localisée et fluide.

6 Résultats et Discussion

L'application développée représente une solution complète et modulaire pour l'analyse et la gestion des avis clients. L'ensemble des fonctionnalités, de la collecte des données à leur visualisation sur une interface conviviale, illustre une intégration cohérente entre différentes composantes techniques. L'utilisation de Docker a permis de garantir une portabilité et une configuration simplifiée de l'application, réduisant considérablement les problèmes liés à l'environnement d'exécution. La base de données SQLite joue un rôle central en assurant un stockage structuré et efficace des informations, facilitant ainsi les interactions avec les modules d'analyse.

Le pipeline de traitement des avis, bien qu'étant un élément clé, s'inscrit dans un cadre plus large qui inclut la récupération automatisée des données des restaurants, leur enrichissement par des analyses NLP, et l'amélioration de l'expérience avec *Mistral*. Les résumés produits par les analyses NLP offrent une vue d'ensemble des tendances et opinions exprimées par les clients, contribuant ainsi à une meilleure compréhension des attentes des utilisateurs finaux.

En termes d'expérience utilisateur, l'application basée sur Streamlit propose une interface interactive et intuitive, permettant une exploration en temps réel des résultats. Les utilisateurs peuvent accéder rapidement aux informations pertinentes sur les restaurants, leurs avis et les tendances générales. Cette fluidité d'interaction représente un atout majeur dans un contexte où la rapidité et la simplicité d'accès aux données sont primordiales.

Cependant, certains défis subsistent. La qualité des résultats est fortement dépendante de celle des données initiales, notamment pour les avis comportant des fautes ou un langage informel. De plus, le traitement d'un volume important d'avis peut entraîner des ralentissements, bien que cela puisse être atténué par des optimisations futures. D'autres limitations concernent la dépendance aux modèles utilisés pour les résumés, dont la pertinence peut varier en fonction de la complexité des avis analysés.

Malgré ces limitations, cette application se distingue par sa modularité et sa capacité d'adaptation, offrant de nombreuses opportunités d'amélioration. À l'avenir, des ajouts tels que des modèles plus avancés de détection des sentiments ou des tableaux de bord plus détaillés pourraient enrichir davantage les fonctionnalités et répondre à des besoins spécifiques d'analyse. En conclusion, cette application démontre un équilibre entre efficacité, évolutivité et accessibilité, offrant une base solide pour le traitement des avis clients et la valorisation des données dans le secteur de la restauration.

Conclusion

Ce projet a permis de développer une application complète et modulable pour l'analyse et la gestion des avis clients dans le secteur de la restauration. En intégrant des technologies modernes telles que Docker, SQLite et Streamlit, ainsi que des techniques avancées de traitement du langage naturel, l'application offre une solution efficace pour collecter, traiter et présenter les données de manière intuitive. Les résultats obtenus démontrent la capacité de l'application à extraire des informations pertinentes et à fournir des résumés synthétiques qui facilitent la prise de décision. Bien que des défis subsistent, notamment liés à la qualité des données et aux performances, ce projet constitue une base solide pour des améliorations futures et des extensions possibles. Ainsi, cette application illustre les opportunités offertes par la combinaison de l'intelligence artificielle et des outils numériques pour valoriser les données dans un contexte professionnel.

Références

[1] TensorFlow Documentation. (n.d.). Retrieved from `https://www.tensorflow.org/`

[2] Keras Documentation. (n.d.). Retrieved from `https://keras.io/`

[3] Scikit-learn Documentation. (n.d.). Retrieved from `https://scikit-learn.org/stable/`

[4] SQLite Documentation. (n.d.). Retrieved from `https://www.sqlite.org/docs.html`

[5] Python Documentation. (n.d.). Retrieved from `https://docs.python.org/3/`

[6] Streamlit Documentation. (n.d.). Retrieved from `https://docs.streamlit.io/`

[7] Docker Documentation. (n.d.). Retrieved from `https://docs.docker.com/`

[8] Mistral AI. (n.d.). Retrieved from `https://mistral.ai/`

Annexes

L'interface ci-dessous permet de configurer l'adresse de l'utilisateur, ce qui est une étape importante pour bénéficier pleinement des fonctionnalités de l'application.

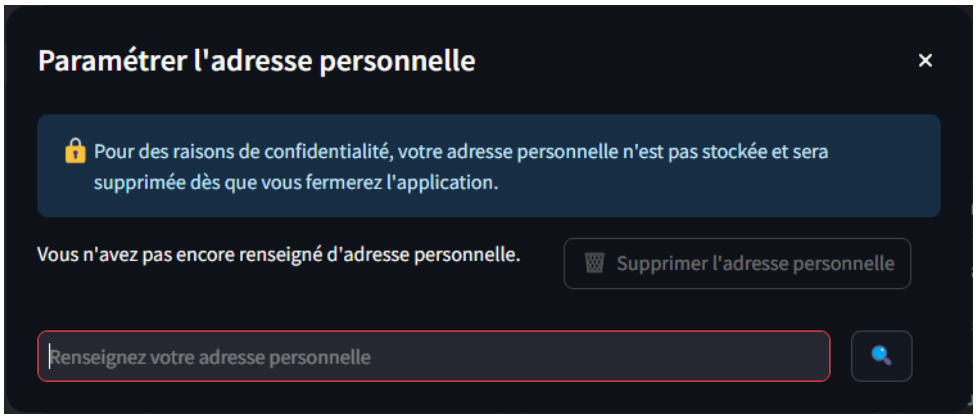

FIGURE 5 – Adresse - Page Accueil

La page Explorer permet d'accéder aux fonctionnalités clés de l'application.

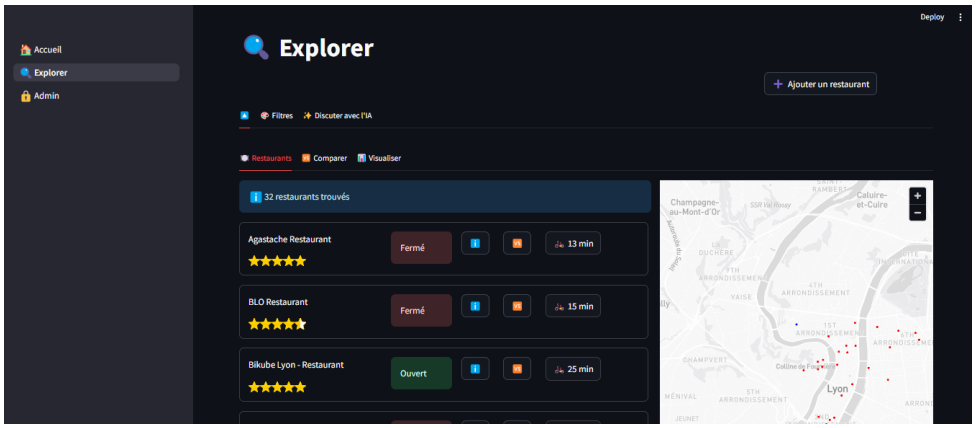

FIGURE 6 – Liste des restaurant - Page Explorer

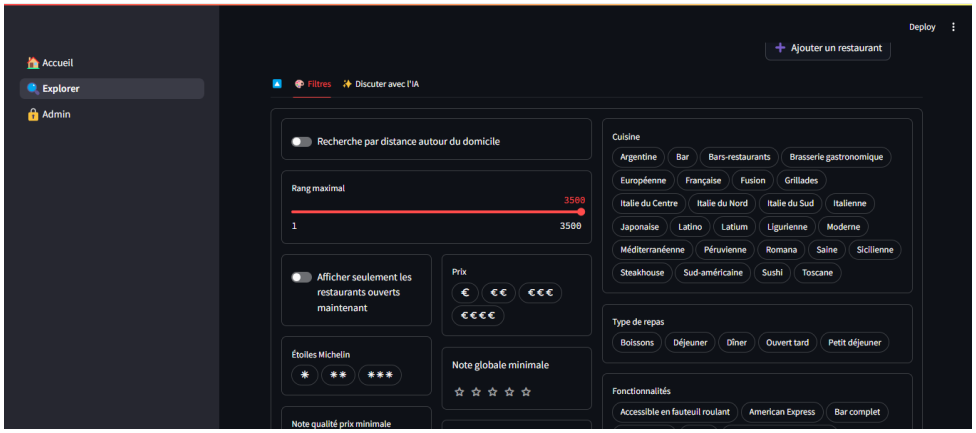

FIGURE 7 – Différents filtres sur les restaurants

Dans le comparateur, l'utilisateur peut comparer jusqu'à trois restaurant simultanément.

Sur l'image ci-dessous, nous avons la page de présentation du restaurant Agastache Restaurant. Il est accessible en cliquant sur l'icône Info dans la liste des restaurants. Dans

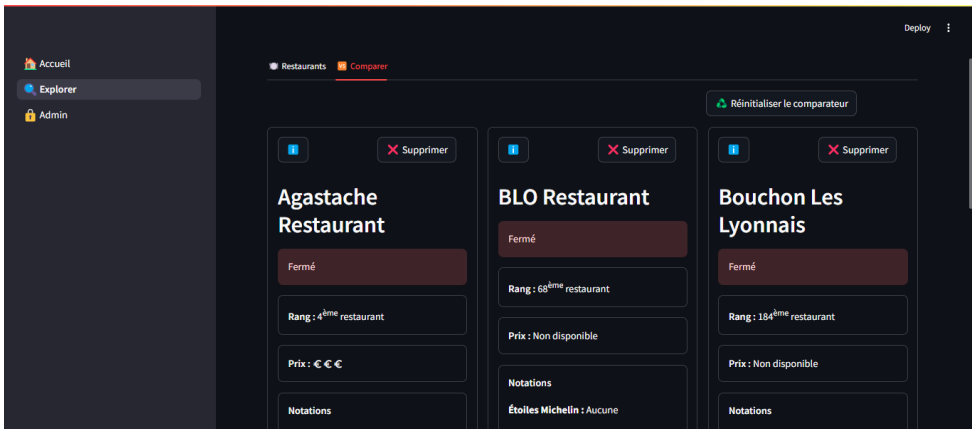

FIGURE 8 – Comparateur de restaurant

cet onglet, nous avons les informations générales sur le restaurant (coordonnées, horaires d'ouverture, cuisines, notes, ...).

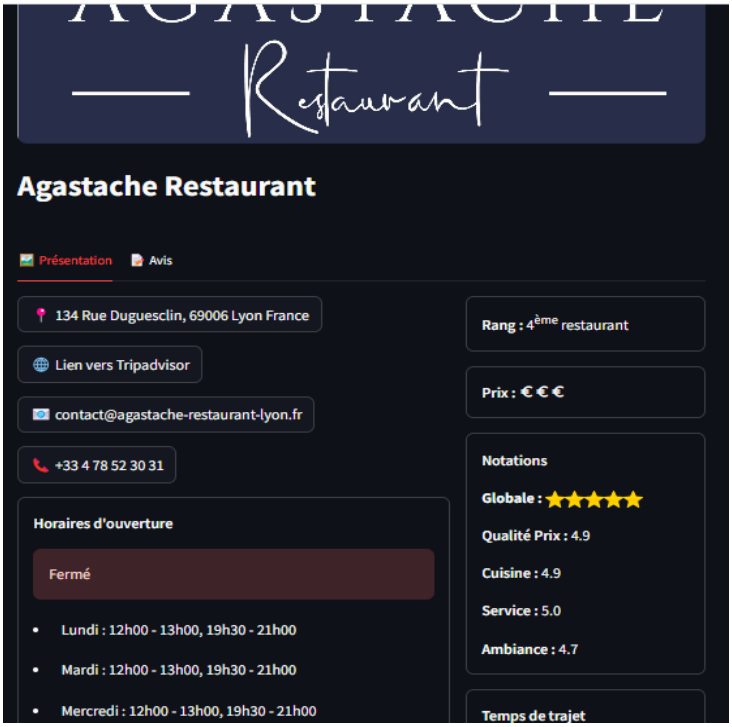

FIGURE 9 – Présentation d'un restaurant - Agastache Restaurant

Dans l'onglet Avis, nous avons recensé les avis, les top contributeurs. Nous avons affiché la répartition notes, les types de visites et l'évolution des avis dans le temps.

FIGURE 10 – Avis d'un restaurant - Agastache Restaurant

Sur la page Admin, l'utilisateur peut administrer la base de données avec des requêtes simples impliquant ajout de données, suppression, modification et même une sélection. Il est aussi possible de faire des jointures entre différentes tables.

FIGURE 11 – Administration