Projet de Collecte de Données depuis GitHub

Date de réalisation: 12/06/23 - 17/06/23

Introduction :

Ce document présente les étapes réalisées et les difficultés rencontrées lors de la réalisation du projet de collecte de données depuis GitHub. L'objectif était de collecter des informations pertinentes sur les dépôts de code hébergés sur GitHub pour mieux comprendre les tendances de développement, les langages de programmation les plus utilisés et identifier des projets intéressants.

Définition :

Le web scraping fait simplement référence à l'extraction de données à partir d'un site web.

Les différentes méthodes de collecte de données à partir de GitHub :

Utilisation de l'API GitHub :

- Avantages :
  - Accès direct aux fonctionnalités de GitHub.
  - Données actualisées en temps réel.
  - Flexibilité dans la récupération des informations.
- Limitations :
  - Limites de fréquence possibles.
  - Authentification requise pour certaines fonctionnalités.

Utilisation de bibliothèques Python comme PyGitHub :

- Avantages :
  - Interface conviviale pour interagir avec l'API GitHub.
  - Fonctionnalités supplémentaires pour simplifier les tâches courantes.
- Limitations :
  - Dépendance externe à installer et à importer.
  - Mises à jour régulières nécessaires.

Avant de collecter des données depuis GitHub :

1-Vérifiez les termes de service : Consultez les termes de service du site web pour savoir s'ils autorisent ou interdisent explicitement le web scraping. Respectez les restrictions et politiques énoncées.

2-Recherchez un fichier robots.txt : Consultez le fichier robots.txt du site web pour voir s'il autorise ou interdit le web scraping. Respectez les directives spécifiées dans ce fichier.

3-Identifiez les lois et réglementations : Informez-vous sur les lois et réglementations relatives au web scraping dans votre pays ou juridiction. Assurez-vous de respecter les droits d'auteur, la confidentialité des données et autres obligations légales.

4-Considérez l'éthique et le respect de la propriété intellectuelle : Veillez à ne pas violer les droits de propriété intellectuelle du site web que vous scrapez. Respectez les droits d'auteur et les restrictions d'utilisation des données.

5-Limitez les demandes et utilisez des délais appropriés : Évitez de surcharger le serveur du site web en envoyant trop de requêtes de scraping en peu de temps. Respectez les délais recommandés et limitez la fréquence de vos requêtes.

6-Utilisez des bibliothèques et outils appropriés : Utilisez des bibliothèques et des outils de scraping reconnus pour vous assurer de respecter les bonnes pratiques et éviter les problèmes juridiques.

7-Contactez le propriétaire du site web : Si vous avez des doutes ou des questions concernant l'autorisation de faire du web scraping, contactez le propriétaire du site web pour obtenir des informations supplémentaires ou demander une autorisation formelle.

Script Python :

Ce code est un script Python qui utilise l'API GitHub pour collecter des informations sur les dépôts créés entre une date de début et une date de fin spécifiées par l'utilisateur. Il récupère les détails tels que le nom du dépôt, l'URL, la description, le nombre d'étoiles, la date de création, le langage utilisé, le nombre de forks, le nombre de watchers, le nombre d'issues ouvertes et le nom du propriétaire.
Le script utilise une approche de pagination pour récupérer un certain nombre de dépôts par jour, conformément à la valeur fournie par l'utilisateur. Il gère également les erreurs de requête en cas de dépassement des limites de l'API. Une fois les données collectées, le script les écrit dans un fichier CSV spécifié par l'utilisateur, avec des en-têtes correspondant aux informations extraites.
En fin d'exécution, le script affiche un message indiquant que les dépôts ont été collectés avec succès et enregistrés dans le fichier CSV.
Le script Python est disponible sur : Faissal-00/Scraping_GitHub

Étapes réalisées :

Étape 1: Définition des objectifs du projet

-Compréhension des besoins de l'entreprise en termes d'informations sur les dépôts GitHub
-Identification des informations pertinentes à collecter

Étape 2: Recherche et documentation

-Analyse de la structure de l'API GitHub et des fonctionnalités disponibles
-Documentation sur les bonnes pratiques de collecte de données depuis des API web

Étape 3: Configuration de l'environnement de développement

-Installation de Python et des bibliothèques nécessaires (requests, datetime, csv)

Étape 4: Collecte de données depuis l'API GitHub

-Mise en place d'une stratégie de gestion des erreurs et des requêtes (retry mechanism)
-Authentification avec un access token pour augmenter les limites de requêtes
-Interaction avec l'API GitHub pour récupérer les dépôts créés sur une période donnée et avec un certain nombre de repositories par jour
-Extraction des informations pertinentes (nom, URL, description, étoiles, date de création, langage, forks, watchers, open issues, propriétaire)
-Pagination pour récupérer tous les dépôts correspondants aux critères

Étape 5: Écriture des données collectées dans un fichier CSV

-Création d'un fichier CSV pour stocker les informations des dépôts collectées
-Définition des en-têtes du fichier CSV
-Écriture des données collectées dans le fichier CSV

Difficultés rencontrées lors de la collecte de données :

Au cours de la collecte de données depuis GitHub, j'ai fait face à plusieurs difficultés qui ont nécessité des ajustements et des solutions. Voici les problèmes que j'ai rencontrés :

-Limite de l'API à 1000 dépôts : Lors de l'utilisation de l'API GitHub, j'ai découvert que la limite de requêtes par page est fixée à 1000 dépôts. Cela signifie que je ne pouvais récupérer que les 1000 premiers dépôts

correspondant à mes critères de recherche. Pour remédier à cela, j'ai dû mettre en place une pagination adéquate et ajuster mes paramètres de recherche pour obtenir les résultats les plus pertinents.

-Gestion des doublons de données : Lors de la collecte initiale de 20 000 dépôts, j'ai constaté que certains dépôts étaient en doublon. Après avoir supprimé les doublons, j'ai constaté qu'il ne restait que 139 dépôts uniques. Cela m'a amené à ajuster mon approche et à mettre en place une méthode de déduplication directement dans le code lors de la collecte des données.

- Objectif de suppression des doublons lors de la collecte : Initialement, mon objectif était de supprimer les doublons pendant la collecte des données. Cependant, en raison de certaines contraintes techniques, j'ai dû collecter 5 000 dépôts et gérer la suppression des doublons dans le code après coup. Cela a été un défi supplémentaire pour garantir la précision et l'exhaustivité des données collectées.

Conclusion :

Le projet de collecte de données depuis GitHub a été réalisé avec succès. Les étapes de collecte des données ont été mises en place en utilisant l'API.