jumia_product_scraper.py

Vue d'ensemble

JumiaDataExtractor est un outil de scraping conçu pour extraire des données de produits depuis le site e-commerce Jumia. Il permet de récupérer systématiquement les informations des produits à travers différentes catégories et sous-catégories.

Prérequis

- Python 3.x
- Bibliothèques requises :
    - BeautifulSoup4
    - Requests
    - JSON
    - CSV
    - Os
    - Datetime
    - Time

Structure du Projet

Organisation des Dossiers

base_directory/

├── jumia_data/

│ ├── categories/

│ │ └── all_categories.json

│ └── subcategories/

│ └── processed_subcategories.csv

└── data/

  └── [fichiers de sortie CSV]

Fonctionnalités Principales

1. Initialisation (`__init__`)

- Configure les chemins des dossiers et fichiers
- Crée la structure de dossiers nécessaire
- Vérifie l'existence des fichiers requis

2. Conversion JSON vers CSV (`convert_json_to_csv`)

- Convertit les données de catégories du format JSON vers CSV
- Traite les sous-catégories
- Génère un fichier CSV structuré avec les champs : text, href, subcategory

3. Extraction des Produits (`scrape_category`)

- Récupère les produits page par page pour chaque catégorie
- Gère la pagination automatiquement
- Extrait les détails suivants pour chaque produit :
    - Nom
    - Prix
    - Marque
    - Catégorie
    - ID du produit
    - Prix affiché
    - URL de l'image
    - URL du produit
    - Sous-catégorie
    - Nom de la catégorie
    - Numéro de page
    - URL de la page

4. Gestion des URLs (`clean_url`, `get_page_url`)

- Nettoie les URLs pour assurer une pagination correcte
- Génère les URLs appropriées pour chaque page

5. Sauvegarde des Données (`save_data`)

- Sauvegarde les données extraites en format CSV
- Ajoute un horodatage aux noms de fichiers

- Crée des fichiers séparés pour chaque catégorie
- Génère un fichier global contenant tous les produits

Utilisation

Exécution Basique

```
extractor = JumiaDataExtractor()

extractor.process_all_categories()
```

Flux de Travail

1. Initialisation de l'extracteur
2. Conversion des données JSON en CSV
3. Traitement séquentiel des catégories
4. Extraction des produits
5. Sauvegarde des données

Gestion des Erreurs

- Vérification de l'existence des fichiers requis
- Gestion des erreurs de requêtes HTTP
- Traitement des exceptions lors de l'extraction
- Messages d'erreur détaillés avec traçage

Bonnes Pratiques Implémentées

1. Délai entre les requêtes (2 secondes)
2. Utilisation d'en-têtes User-Agent
3. Gestion propre des chemins de fichiers
4. Structure de données cohérente
5. Sauvegarde progressive des données

Limitations et Considérations

- Dépend de la structure HTML de Jumia
- Nécessite une connexion Internet stable
- Sensible aux changements d'interface du site
- Les performances dépendent du nombre de produits et de pages

Format des Données de Sortie

Les fichiers CSV générés contiennent les champs suivants :

name, price, brand, category, product_id, displayed_price, image_url, product_url, subcategory, category_name, page_number, page_url

Messages de Log

Le script fournit des informations détaillées sur :

- L'initialisation des dossiers
- La progression du scraping
- Le nombre de pages et de produits
- Les erreurs éventuelles
- La sauvegarde des fichiers

Maintenance et Mises à Jour

Pour maintenir le script fonctionnel :

1. Vérifier régulièrement la structure HTML de Jumia
2. Adapter les sélecteurs CSS si nécessaire
3. Mettre à jour les en-têtes HTTP si requis
4. Ajuster les délais entre requêtes selon les besoins

Documentation JumiaScraper

Vue d'ensemble

JumiaScraper est un outil spécialisé dans l'extraction des catégories et sous-catégories du site e-commerce Jumia. Il construit une hiérarchie complète des catégories de produits disponibles sur le site.

Prérequis

- Python 3.x
- Bibliothèques requises :
    - Requests
    - BeautifulSoup4
    - JSON
    - Os
    - Time

Structure du Projet

Organisation des Dossiers

```
base_directory/
└── jumia_data/
    ├── elements/
    │   └── elements.json
    ├── categories/
    │   └── all_categories.json
    └── subcategories/
        └── [catégorie]_subcategories.json
```

Configuration Initiale

Variables d'Environnement

base_url = "https://www.jumia.ma"
start_url = "https://www.jumia.ma/telephone-tablette/n"

En-têtes HTTP

headers = {

```
    'User-Agent': 'Mozilla/5.0...',
    'Accept': 'text/html,application/xhtml+xml...',
    'Accept-Language': 'fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3'
}
```

Fonctionnalités Principales

1. Gestion des Dossiers (`create_directory`)

- Crée la structure de dossiers nécessaire
- Vérifie l'existence des dossiers avant création
- Gère trois sous-dossiers principaux :
    - elements
    - categories
    - subcategories

2. Sauvegarde JSON (`save_json`)

- Sauvegarde les données au format JSON
- Crée automatiquement les dossiers manquants
- Utilise un encodage UTF-8
- Formate le JSON avec indentation
- Gère les caractères non-ASCII

3. Extraction du Contenu (`get_page_content`)

- Effectue les requêtes HTTP
- Utilise des en-têtes personnalisés
- Parse le HTML avec BeautifulSoup
- Gère les erreurs de connexion

4. Extraction des Éléments (`extract_elements`)

- Cible les éléments avec la classe "-db -pvs -phxl -hov-bg-gy05"
- Extrait pour chaque élément :
    - Texte
    - HTML brut
    - Lien (href)

5. Scraping Principal (`scrape_content`)

- Processus en plusieurs étapes :
    1. Création des dossiers
    2. Scraping de la page principale
    3. Extraction des éléments principaux
    4. Scraping des sous-catégories
    5. Sauvegarde des données

Format des Données

Structure des Éléments Principaux

```
headers = {
   'User-Agent': 'Mozilla/5.0...',
   'Accept': 'text/html,application/xhtml+xml...',
   'Accept-Language': 'fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3'
}
```

Structure des Catégories Complètes

```
{
   "category": "Nom de la catégorie",
   "url": "URL complète",
   "sub_elements": [
      {
         "text": "Nom de la sous-catégorie",
         "html": "Code HTML brut",
         "href": "Lien relatif"
      }
   ]
}
```

Gestion des Erreurs

- Vérification des réponses HTTP
- Capture des exceptions lors des requêtes
- Traçage détaillé des erreurs
- Messages d'erreur informatifs

Bonnes Pratiques Implémentées

1. Délai entre les requêtes (1 seconde)
2. En-têtes HTTP complets
3. Gestion propre des chemins de fichiers
4. Nettoyage des noms de fichiers
5. Sauvegarde progressive des données

Utilisation

Exécution Basique

```
scraper = JumiaScraper()
scraper.scrape_content()
```

Flux d'Exécution

1. Initialisation du scraper
2. Création des dossiers
3. Extraction des catégories principales
4. Traitement des sous-catégories
5. Sauvegarde des données

Limitations et Considérations

- Dépend de la structure HTML de Jumia
- Sensible aux changements d'interface
- Requiert une connexion Internet stable
- Spécifique à la section "telephone-tablette"

Logs et Suivi

Le script fournit des informations sur :

- L'initialisation des dossiers
- La progression du scraping
- Les URLs traitées
- Le nombre d'éléments extraits
- Les erreurs éventuelles
- La sauvegarde des fichiers

Maintenance

Pour maintenir le script :

1. Vérifier régulièrement la structure HTML
2. Mettre à jour les en-têtes HTTP si nécessaire
3. Adapter les sélecteurs CSS si besoin
4. Ajuster les délais entre requêtes

Documentation JumiaDataOrganizer

Vue d'ensemble

JumiaDataOrganizer est un outil conçu pour organiser, traiter et archiver les données extraites de Jumia. Il gère la structuration quotidienne des données et l'archivage automatique des anciennes données.

Prérequis

- Python 3.x
- Bibliothèques requises :
    - Pandas
    - Os
    - Datetime
    - Shutil
    - Glob

Structure du Projet

Organisation des Dossiers

```
base_directory/
├── data/
│   └── all_products_*.csv
└── daily_data/
    ├── YYYYMMDD/
    │   ├── jumia_data_YYYYMMDD.csv
    │   └── category_*_YYYYMMDD.csv
    └── archives/
        └── YYYYMMDD_archive.zip
```

1. Initialisation (`__init__`)

- Configure les chemins des dossiers
- Crée la structure de dossiers nécessaire
- Initialise les chemins :
    - data_directory : stockage des données brutes
    - daily_data_path : stockage des données organisées par jour

2. Gestion des Fichiers Sources (`get_latest_all_products_file`)

- Recherche le fichier all_products le plus récent
- Utilise le pattern "all_products_*.csv"
- Trie les fichiers par date de création
- Retourne le chemin du fichier le plus récent

3. Organisation Quotidienne (`process_new_data`)

Fonctionnalités :

- Lecture du fichier source le plus récent
- Création d'un dossier daté (YYYYMMDD)
- Sauvegarde du fichier complet
- Séparation par catégories
- Nettoyage des noms de fichiers

Structure des fichiers générés :

jumia_data_YYYYMMDD.csv # Données complètes

category_[nom]_YYYYMMDD.csv # Données par catégorie

4. Archivage (`archive_old_data`)

- Archive les données plus anciennes que X jours (défaut: 30)
- Crée des archives ZIP datées
- Supprime les dossiers originaux après archivage
- Maintient une structure d'archives organisée

Traitement des Données

Processus de Nettoyage

1. Lecture des données source
2. Vérification des colonnes
3. Traitement des valeurs NaN
4. Nettoyage des noms de catégories

Format des Fichiers

- Entrée : CSV avec colonnes incluant 'category'
- Sortie :

    - Fichier principal quotidien
    - Fichiers par catégorie
    - Archives ZIP pour les anciennes données

Gestion des Erreurs

- Vérification de l'existence des fichiers
- Gestion des formats de date invalides
- Traitement des valeurs manquantes
- Messages d'erreur détaillés avec traçage

Utilisation

Exécution Basique

```
organizer = JumiaDataOrganizer()
organizer.process_new_data()
organizer.archive_old_data(days_to_keep=30)
```

Flux de Travail

1. Initialisation de l'organisateur
2. Traitement des nouvelles données
3. Organisation en dossiers quotidiens
4. Archivage des anciennes données

Configuration

Paramètres Modifiables

- `days_to_keep` : Nombre de jours avant archivage
- Chemins des dossiers
- Format des noms de fichiers

Structure des Dossiers

- `/data` : Données brutes
- `/daily_data` : Données organisées
- `/daily_data/archives` : Archives ZIP

Bonnes Pratiques Implémentées

1. Organisation chronologique des données
2. Nettoyage automatique des noms de fichiers
3. Archivage systématique
4. Gestion des erreurs robuste
5. Traçabilité des opérations

Limitations et Considérations

- Requiert un espace disque suffisant
- Dépend de la structure des fichiers source
- Sensible aux formats de date
- Performance dépendante de la taille des données

Maintenance

Pour maintenir le script :

1. Vérifier régulièrement l'espace disque
2. Ajuster la période d'archivage si nécessaire
3. Nettoyer périodiquement les archives
4. Mettre à jour les formats de fichiers si besoin

Logs et Suivi

Le script fournit des informations sur :

- L'initialisation des dossiers
- Le traitement des fichiers
- La création des archives
- Les erreurs éventuelles

Documentation DAG Airflow - Jumia Price Tracker

Vue d'ensemble

Ce DAG (Directed Acyclic Graph) Airflow automatise le processus de suivi des prix sur Jumia en orchestrant l'extraction, le traitement et l'organisation des données de produits.

Configuration du DAG

Informations de Base

```
dag_id: 'jumia_price_tracker'
schedule_interval: '0 12 * * *'  # Tous les jours à midi
start_date: 2025-02-16
tags: ['jumia', 'scraping', 'prices']
```

Arguments par Défaut

```
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5)
}
```

Structure du Pipeline

1. Extraction des Catégories (Task 1)

Task ID: `extract_categories`

Fonctionnalité:

- Initialise JumiaScraper
- Extrait toutes les catégories et sous-catégories
- Sauvegarde les données en JSON
- Vérifie le succès de l'extraction

Gestion des Erreurs:

- Lève une exception en cas d'échec
- Retourne un message de succès

2. Extraction des Produits (Task 2)

Task ID: `extract_products`

Fonctionnalité:

- Initialise JumiaDataExtractor
- Convertit les données JSON en CSV
- Extrait les produits de chaque catégorie
- Sauvegarde les données en CSV

Étapes:

1. Conversion JSON vers CSV
2. Extraction des produits
3. Vérification des données extraites

3. Organisation des Données (Task 3)

Task ID: `organize_data`

Fonctionnalité:

- Initialise JumiaDataOrganizer
- Organise les données par date
- Archive les données anciennes
- Maintient la structure des dossiers

Paramètres:

- `days_to_keep`: 30 jours avant archivage

Dépendances des Tâches

extract_categories >> extract_products >> organize_data

Modules Requis

from airflow import DAG
from airflow.operators.python import PythonOperator

```
from datetime import datetime, timedelta
from scripts.jumia_category_scraper import JumiaScraper
from scripts.jumia_product_scraper import JumiaDataExtractor
from scripts.jumia_data_organizer import JumiaDataOrganizer
```

Documentation des Tâches

Extract Categories

Extraction des catégories
Cette tâche extrait toutes les catégories et sous-catégories de Jumia.
Les données sont sauvegardées au format JSON.

Extract Products

Extraction des produits
Cette tâche:
1. Convertit les données JSON en CSV
2. Extrait tous les produits de chaque catégorie
3. Sauvegarde les données au format CSV

Organize Data

Organisation des données
Cette tâche:
1. Organise les données par date et catégorie
2. Archive les anciennes données
3. Maintient une structure de dossiers propre

Gestion des Erreurs

Mécanismes de Retry

- Nombre de tentatives: 1
- Délai entre tentatives: 5 minutes
- Notification par email en cas d'échec

Points de Vérification

1. Succès de l'extraction des catégories

2. Conversion JSON vers CSV réussie
3. Présence de produits extraits
4. Succès de l'organisation des données

Surveillance et Maintenance

Logs

Chaque tâche génère des logs détaillés sur:

- Début d'exécution
- Progression
- Succès/Échec
- Erreurs éventuelles

Points d'Attention

1. Vérifier l'espace disque disponible
2. Surveiller les temps d'exécution
3. Monitorer les échecs d'archivage
4. Vérifier les notifications email

Bonnes Pratiques

1. Documentation inline avec doc_md
2. Gestion des erreurs à chaque étape
3. Structure modulaire des tâches
4. Paramètres configurables
5. Messages de log informatifs

Limitations et Considérations

- Dépend de la disponibilité du site Jumia
- Sensible aux changements d'interface
- Requiert une gestion de l'espace disque
- Impact potentiel des timeouts Airflow

Maintenance

Pour maintenir le DAG:

1. Vérifier régulièrement les logs
2. Ajuster les paramètres de retry si nécessaire
3. Mettre à jour les scripts sous-jacents
4. Surveiller l'utilisation des ressources