



Université Chouaib Doukkali

Faculté des Sciences – El Jadida

جامعة شعيب الدكالي

Université Chouaib Doukkali

Projet de fin de module

Analyse des Publications Scientifiques

avec l'API Scopus

Elaboré Par:

- Saadaoui hajar
- Elbouzidi Soukaina
- Ennasser Kawtar

Encadré Par:

- Pr. Madani

Sommaire

Introduction général	1
I- Cadre du projet	2
1. Contexte du projet	2
2. Objectif du projet	2
II. Méthodologie	2
1. Inscription et Configuration de l'API Scopus	2
1.1. Inscription	2
1.2. Obtention de la Clé API	3
1.3. Familiarisation	4
III. Réalisation	4
1. Préparation de l'Environnement de Développement	4
2. Récupération des Données	4
2.1 Récupération des données à l'aide de la clé API	5
2.2 Récupération des données spécifiques à partir de leurs DOIs	6

3. Analyse des Données	8
4 Visualisation des Données	19
5. RDF et SPARQL	27
Conclusion	36

Introduction générale

Dans le cadre de notre projet de fin de module , nous avons entrepris un projet visant à analyser les publications scientifiques à l'aide de l'API Scopus. Scopus est une vaste base de données bibliographiques gérée par Elsevier, couvrant une large gamme de disciplines scientifiques, techniques, médicales et sociales. L'objectif principal de ce projet est de récupérer des informations sur les publications scientifiques à partir de Scopus, puis d'analyser et de visualiser ces données en utilisant des bibliothèques Python telles que NumPy, Pandas et Matplotlib.

L'API Scopus fournit un accès programmatique aux données de la base de données, permettant ainsi d'automatiser la récupération des publications et de leurs métadonnées. Ce projet nous offre l'opportunité de nous familiariser avec l'utilisation des APIs pour extraire des données réelles, de les manipuler et de les analyser avec des outils puissants, et de les visualiser pour mieux comprendre les tendances et les distributions.

Ce rapport détaille les étapes suivies pour réaliser ce projet, de l'inscription à l'API Scopus jusqu'à l'analyse et la visualisation des données récupérées. Nous présenterons également les résultats obtenus et les conclusions tirées de notre analyse.

I. Cadre du projet:

1. Contexte du projet:

Le domaine de la recherche scientifique est en constante expansion, avec des milliers de nouvelles publications chaque jour. Les chercheurs, étudiants et professionnels ont besoin d'outils efficaces pour explorer, analyser et visualiser ces vastes ensembles de données. Scopus, en tant que l'une des plus grandes bases de données bibliographiques, offre un accès à une mine d'informations. En utilisant l'API Scopus, nous pouvons automatiser la récupération des publications et de leurs métadonnées, ce qui facilite une analyse approfondie et des visualisations significatives des tendances et des distributions des publications scientifiques.

2. Objectifs du projet:

Les objectifs spécifiques de ce projet incluent :

1. **Récupération des Données** : Utiliser l'API Scopus pour extraire des informations sur les publications scientifiques.
2. **Analyse des Données** : Utiliser des bibliothèques Python (NumPy et Pandas) pour manipuler et analyser les données récupérées.
3. **Visualisation des Données** : Utiliser Matplotlib pour créer des graphiques et visualiser les résultats de l'analyse.

II. Méthodologie:

Dans cette section, nous décrivons les méthodes et les étapes que nous avons suivies pour atteindre les objectifs du projet.

1. Inscription et Configuration de l'API Scopus:

1.1. Inscription :

Nous avons commencé par accéder au site "Elsevier Developer Portal" où nous avons entamé la création d'un compte en cliquant sur "Register". Après avoir renseigné le formulaire d'inscription avec nos détails personnels, y compris notre email et notre mot de passe, nous avons finalisé le processus en validant notre compte via le lien envoyé par email par Elsevier.

1.2. Obtention de la Clé API:

Obtention de la clé API nécessaire pour accéder aux données de Scopus.
Une fois connectés à notre compte, nous avons accédé à la section "API Keys".

The screenshot shows the Elsevier Developer Portal homepage. At the top, there's a navigation bar with links for "My API Key", "FAQ", "Products", "Documentation", "Start Coding", and "Contact Us". Below the navigation, the main content area is titled "Elsevier Research Products APIs". It contains a list of options for researchers: "Non-Commercial Users (Researchers in Academic & Public Sector Institutions, Charities & Charitable Foundations)" and "Commercial Users (Researchers in Private Sector & Commercial Institutions)". Below these are three numbered sections: "1. Request an API Key", "2. Look at use cases", and "3. Start coding". Each section has a brief description and a button: "I want an API Key" (for section 1), "Use cases" (for section 2), and "How to Guides" (for section 3). At the bottom, there are links for "About the APIs" and "Get programmatic access to:".

- Nous avons cliqué sur "My API Key", puis sur "Create API Key", et suivi les instructions pour générer une nouvelle clé API.

The screenshot shows the "My API Key" page of the Elsevier Developer Portal. The top navigation bar is identical to the homepage. The main content area is titled "Registered API keys". It displays a table with two rows of data:

#	Website URL	Label	API Key
1			6abe81ebffffe2691e33795aec177fe9
2			9aebde1fa88b0b7325c7d8054dd3e754

On the right side of the table, there is a "Create API Key" button. At the bottom of the page, there's a footer with links for "Support", "Terms and conditions", "Privacy policy", and "Cookie policy", along with the Elsevier logo and copyright information.

1.3. Familiarisation:

Nous avons commencé par accéder à la documentation complète de l'API Scopus, où nous avons minutieusement étudié les divers types de requêtes disponibles, les paramètres requis pour les requêtes, ainsi que les formats de réponse attendus. En explorant les exemples de requêtes fournis, nous avons acquis une meilleure compréhension de la manière de structurer efficacement nos propres appels API. De plus, nous avons pris soin de noter les limitations spécifiques et les bonnes pratiques recommandées pour garantir une utilisation correcte et optimale de l'API tout au long du projet.

III. Réalisation:

1. Préparation de l'Environnement de Développement:

Dans cette étape, nous avons configuré notre environnement de développement Python en installant les bibliothèques nécessaires à l'aide de pip. Ces bibliothèques sont essentielles pour interagir avec l'API Scopus, manipuler les données récupérées, et créer des visualisations graphiques à l'aide de Python. Les bibliothèques installées incluent requests pour la gestion des requêtes HTTP, pandas pour la manipulation et l'analyse des données, numpy pour les calculs scientifiques avancés, et matplotlib pour la création de graphiques et de visualisations.

```
pip install requests pandas numpy matplotlib
Note: you may need to restart the kernel to use updated packages. Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: requests in c:\programdata\anaconda3\lib\site-packages (2.31.0)
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (2.1.4)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (1.26.4)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (3.8.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\programdata\anaconda3\lib\site-packages (from requests) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib\site-packages (from requests) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\programdata\anaconda3\lib\site-packages (from requests) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-packages (from requests) (2024.2.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

DEPRECATION: Loading egg at c:\programdata\anaconda3\lib\site-packages\vboxapi-1.0-py3.11.egg is deprecated. pip 24.3 will enforce this behaviour change.
A possible replacement is to use pip for package installation.. Discussion can be found at https://github.com/pypa/pip/issues/12330
```

2. Récupération des Données:

Après avoir configuré notre accès à l'API Scopus, nous avons entrepris la récupération des données nécessaires pour notre projet. À l'aide de Python et de la bibliothèque requests, nous avons écrit un script pour interagir avec l'API Scopus et récupérer les informations sur les publications scientifiques. Voici les étapes clés que nous avons suivies :

2.1 Récupération des données à l'aide de la clé API :

Pour interagir avec l'API Scopus et récupérer les informations sur les publications scientifiques, nous avons utilisé la bibliothèque requests en Python.

```

import requests
import pandas as pd

def fetch_scopus_data(api_key, query, count=25):
    """Fetch data from the Scopus API."""
    url = 'https://api.elsevier.com/content/search/scopus'
    params = {
        'apiKey': api_key,
        'query': query,
        'count': count
    }

    try:
        # Effectuer la requête GET à l'API Scopus
        response = requests.get(url, params=params)
        response.raise_for_status() # Vérifier s'il y a eu une erreur dans la requête

        # Convertir la réponse JSON en dictionnaire
        data = response.json()

        # Vérifier si les données attendues sont présentes dans la réponse
        if 'search-results' in data and 'entry' in data['search-results']:
            print("La clé API est valide et fonctionne correctement.")
            print(f"Nombre de résultats obtenus : {len(data['search-results']['entry'])}")
            return data['search-results']['entry']
        else:
            print("La structure de la réponse JSON ne contient pas les clés attendues.")
            return None

    except requests.exceptions.HTTPError as http_err:
        print(f'Erreur HTTP {response.status_code}: {response.reason}')
        print(response.text)
    except requests.exceptions.RequestException as req_err:
        print(f'Erreur de requête: {req_err}')
    except Exception as err:
        print(f'Erreur: {err}')

def parse_freetoread(value):
    """Transform the 'freetoread.value' column to readable values."""
    if isinstance(value, list):
        return ', '.join([item['$'] for item in value])
    return value

def clean_and_save_data(entries, filename):
    """Clean the data and save it to a CSV file."""
    if entries:
        # Convertir les entrées JSON en DataFrame Pandas
        df = pd.json_normalize(entries)

        # Nettoyer et organiser les données
        if 'freetoread.value' in df.columns:
            df['freetoread.value'] = df['freetoread.value'].apply(parse_freetoread)

        # Définir les options d'affichage de Pandas pour afficher toutes les lignes et colonnes
        pd.set_option('display.max_rows', None)
        pd.set_option('display.max_columns', None)
        pd.set_option('display.width', None)
        pd.set_option('display.max_colwidth', None)

        # Afficher le DataFrame nettoyé
        print(df)

        # Sauvegarder le DataFrame dans un fichier CSV
        df.to_csv(filename, index=False)
        print(f"Les données ont été nettoyées et sauvegardées dans le fichier {filename}")

# Utiliser les fonctions pour récupérer, nettoyer et sauvegarder les données
api_key = '9aebe1fa88b0b7325c7d8054dd3e754'
query = 'KEY(scopus)'
filename = 'apiscopus_data.csv'

entries = fetch_scopus_data(api_key, query)
clean_and_save_data(entries, filename)

```

Nous avons utilisé une requête GET avec une clé API et une requête de recherche spécifique pour interroger l'API Scopus. La vérification des éléments 'search-results' et 'entry' dans la réponse était essentielle pour confirmer la présence des données attendues. Des blocs try-except ont été inclus pour gérer les erreurs potentielles et assurer la robustesse du script. Après récupération, les données JSON ont été transformées en un DataFrame Pandas avec pd.json_normalize(), facilitant le nettoyage et la manipulation des données. Enfin, le DataFrame nettoyé a été sauvegardé dans le fichier CSV 'api_scopus_data.csv' avec df.to_csv(), assurant une conservation structurée des données pour une analyse future.

2.2 Récupération des données spécifiques à partir de leurs DOIs :

Nous avons utilisé des DOI spécifiques pour interroger l'API Elsevier afin d'enrichir notre ensemble de données. Chaque DOI permet d'accéder à des métadonnées détaillées sur une publication scientifique. Pour ce faire, nous avons développé un script Python utilisant la bibliothèque requests pour envoyer des requêtes HTTP, en intégrant notre clé API Elsevier pour l'authentification. Chaque réponse au format XML a été parsée pour extraire des informations essentielles telles que le titre, les auteurs et les détails de publication. Les données extraites ont été structurées dans un DataFrame Pandas, puis converties en format JSON pour une analyse ultérieure.

```
import requests
import pandas as pd
import xml.etree.ElementTree as ET

# Liste de DOI à récupérer depuis l'API Elsevier
dois = [
    '10.1016/j.cplett.2020.137481',
    '10.1016/j.joule.2020.11.010',
    '10.1016/j.jacc.2020.11.012'
]

# Clé API Elsevier
api_key = '9aebde1fa88b0b7325c7d8054dd3e754'

# Fonction pour récupérer et parser les données d'un DOI spécifique depuis l'API Elsevier
def get_data_from_doi(doi):
    url = f"https://api.elsevier.com/content/article/doi/{doi}"
    headers = {'X-ELS-APIKey': api_key, 'Accept': 'application/xml'}

    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()

        print(f'Statut pour DOI {doi}: {response.status_code}')

        root = ET.fromstring(response.content)
        return parse_xml(root)
    except requests.exceptions.RequestException as e:
        print(f'Error occurred while fetching DOI {doi}: {e}')


def parse_xml(root):
    # Parcourir l'arborescence XML pour extraire les données nécessaires
    # et les structurer en un DataFrame Pandas
    # ...
    pass
```

Programmation python avancée

```
except requests.exceptions.HTTPError as err:
    print(f'Erreur HTTP lors de la récupération du DOI {doi}: {err}')
    return None

except ET.ParseError as e:
    print(f'Erreur de parsing XML pour le DOI {doi}: {e}')
    return None

except Exception as err:
    print(f'Erreur lors de la récupération du DOI {doi}: {err}')
    return None

# Fonction pour parser les données XML et extraire les informations pertinentes
def parse_xml(root):
    namespaces = {
        'dtd': 'http://www.elsevier.com/xml/svapi/article/dtd',
        'dc': 'http://purl.org/dc/elements/1.1/',
        'prism': 'http://prismstandard.org/namespaces/basic/2.0/',
        'xocs': 'http://www.elsevier.com/xml/xocs/dtd'
    }

    data = {
        'doi': root.findtext('.//xocs:doi', namespaces=namespaces),
        'title': root.findtext('.//dc:title', namespaces=namespaces),
        'creator': root.findtext('.//dc:creator', namespaces=namespaces),
        'publicationName': root.findtext('.//prism:publicationName', namespaces=namespaces),
        'volume': root.findtext('.//prism:volume', namespaces=namespaces),
        'issue': root.findtext('.//prism:issueIdentifier', namespaces=namespaces),
        'pageRange': root.findtext('.//prism:pageRange', namespaces=namespaces),
        'coverDate': root.findtext('.//prism:coverDate', namespaces=namespaces),
        'citedby_count': root.findtext('.//xocs:citedby-count', namespaces=namespaces)
    }
    link_elements = root.findall('.//xocs:link', namespaces=namespaces)
    for link in link_elements:
        if link.get('rel') == 'scidir':
            data['doiLink'] = link.get('href')
            break
    return data

# Liste pour stocker les données récupérées depuis les DOI
data_list = []

try:
    # Parcourir la liste des DOI et récupérer les données pour chaque DOI
    for doi in dois:
        data = get_data_from_doi(doi)
        if data:
            data_list.append(data)
    if data_list:
        # Créer un DataFrame à partir des données récupérées
        df_xml = pd.DataFrame(data_list)
        # Convertir le DataFrame en JSON
        df_xml_json = df_xml.to_json(orient='records')
        # Afficher le JSON
        print("\nDataFrame JSON à partir des DOI de l'API Elsevier:")
        print(df_xml_json)
        # Sauvegarder le JSON dans un fichier
        with open('scopus_data_from_dois.json', 'w', encoding='utf-8') as f:
            f.write(df_xml_json)
    else:
        print("Aucune donnée valide n'a été récupérée depuis les DOI.")
except requests.exceptions.HTTPError as err:
    print(f'Erreur HTTP lors de la requête à l\'API Elsevier: {err}')
except Exception as err:
    print(f'Erreur: {err}'')
```

3. Analyse des Données :

Après avoir récupéré les métadonnées à partir des DOI spécifiques à l'aide de l'API Elsevier, nous avons procédé à une analyse approfondie des données pour explorer les caractéristiques et les tendances des publications scientifiques étudiées. Notre approche a permis de capturer des détails essentiels tels que le titre, les auteurs, les détails de publication et le nombre de citations associé à chaque article.

• Nombre Total de Citations

```
import pandas as pd
import matplotlib.pyplot as plt

def analyze_data(df):
    """Analyze the data and provide insights."""
    if df is not None:
        # Calculer le nombre total de citations
        if 'citedby-count' in df.columns:
            df['citedby-count'] = pd.to_numeric(df['citedby-count'], errors='coerce').fillna(0).astype(int)
            total_citations = df['citedby-count'].sum()
            print(f"\nNombre total de citations pour toutes les publications: {total_citations}")

        # Répartition des citations par année de publication
        if 'prism:coverDate' in df.columns:
            df['coverYear'] = pd.to_datetime(df['prism:coverDate'], errors='coerce').dt.year
            citations_per_year = df.groupby('coverYear')['citedby-count'].sum()

        # Affichage des citations par année de publication détaillé
        print("\nCitations par année de publication :")
        for year, citations in citations_per_year.items():
            print(f"Année {year} : {citations} citations")

    # Charger le DataFrame sauvegardé depuis le fichier CSV
    df = pd.read_csv('api_scopus_data.csv')

    # Analyser les données
    analyze_data(df)
```

En agrégeant les données des publications, nous avons identifié un total de 17 citations attribuées à l'ensemble des articles analysés. Ce nombre représente l'accumulation des citations reçues par les publications depuis leur publication jusqu'à la date de récupération des données.

Répartition des Citations par Année de Publication

L'analyse de la répartition des citations par année de publication a révélé les points suivants :

Année 2024 : 17 citations

Année 2025 : 0 citations

```
Nombre total de citations pour toutes les publications: 17
```

```
Citations par année de publication :
Année 2024 : 17 citations
Année 2025 : 0 citations
```

Nombre Total de Publications :

```
def total_publications(df):
    """Calculates the total number of publications."""
    if df is not None:
        total = len(df)
        print(f"\nNombre total de publications dans le dataset : {total}")
    else:
        print("Aucune donnée à analyser.")

# Appel de la fonction
total_publications(df)
```

```
Nombre total de publications dans le dataset : 25
```

Nous avons appliqué une fonction dédiée pour calculer le nombre total de publications incluses dans notre ensemble de données. À partir des DOI sélectionnés, nous avons identifié un total de 25 publications. Cette mesure initiale est essentielle pour établir la base de notre analyse quantitative et qualitative des tendances observées dans les publications scientifiques pertinentes.

- Citations moyennes par publication :

```
def average_citations(df):
    """Calculates the average citations per publication."""
    if 'citedby-count' in df.columns:
        average_citations = df['citedby-count'].mean()
        print(f"\nCitations moyennes par publication : {average_citations:.2f}")
    else:
        print("La colonne 'citedby-count' n'est pas présente dans le DataFrame.")

# Appel de la fonction
average_citations(df)
```

```
Citations moyennes par publication : 0.68
```

Nous avons défini une fonction appelée `average_citations` qui calcule le nombre moyen de citations par publication dans un DataFrame Pandas. Cette fonction commence par vérifier si la colonne 'citedby-count' (qui contient le nombre de citations pour chaque publication) est présente dans le DataFrame passé en paramètre. Si cette colonne existe, la fonction calcule la moyenne des valeurs de cette colonne en utilisant la méthode `mean()` de Pandas et affiche ce résultat avec deux décimales. Si la colonne 'citedby-count' n'est pas présente, la fonction affiche un message d'erreur indiquant que la colonne requise est absente. La fonction est ensuite appelée avec le DataFrame `df` pour effectuer ce calcul sur les données de publications disponibles.

- Identification des Publications les Plus Citées:

```
def publications_most_citations(df, top_n=5):
    """Identifie les publications avec le plus de citations."""
    if 'citedby-count' in df.columns:
        top_publications = df.nlargest(top_n, 'citedby-count')[['dc:title', 'citedby-count']]
        print(f"\nPublications avec le plus de citations (Top {top_n}) :")
        print(top_publications)
    else:
        print("La colonne 'citedby-count' n'est pas présente dans le DataFrame.")

# Charger le DataFrame sauvegardé depuis le fichier CSV
df = pd.read_csv('api_scopus_data.csv')

# Appel de la fonction
publications_most_citations(df)
```

```
Publications avec le plus de citations (Top 5) :

dc:title \
16 Beyond playing 20 questions with nature: Integrative experiment des
ign in the social and behavioral sciences
20 A practical guide to adopti
ng Bayesian analyses in clinical research
0 Influence of phytoplankton, bacteria and víru
ses on nutrient supply in tropical waters
1 Safety and efficacy of low-power pure-cut hot snare polypectomy for small nonpedunculated colorectal polyps compared with conventional resection meth
ods: A propensity score matching analysis
2 Relationship between Self-regulated Learning with Academic Buoyancy: A Case Study among
Malaysia FELDA Secondary School Students

    citedby-count
16          16
20           1
0            0
1            0
2            0
```

Nous avons défini une fonction nommée `publications_most_citations` qui identifie et affiche les publications ayant le plus de citations dans un DataFrame Pandas. La fonction accepte deux arguments : `df`, le DataFrame contenant les données des publications, et `top_n`, un entier indiquant le nombre de publications à afficher (par défaut 5). La fonction commence par vérifier si la colonne '`citedby-count`', qui contient le nombre de citations pour chaque publication, est présente dans le DataFrame. Si cette colonne existe, la fonction utilise la méthode `nlargest` de Pandas pour sélectionner les `top_n` publications ayant le plus de citations, en extrayant les colonnes '`dc:title`' (titre de la publication) et '`citedby-count`' (nombre de citations). Ces publications sont ensuite affichées. Si la colonne '`citedby-count`' n'est pas présente, un message d'erreur est affiché. Avant d'appeler la fonction, le DataFrame `df` est chargé depuis un fichier CSV nommé '`api_scopus_data.csv`', puis la fonction est exécutée pour afficher les publications les plus citées.

Programmation python avancée

• Publications par année :

```
import pandas as pd
import matplotlib.pyplot as plt

def analyze_data(df):
    """Analyse the data and provide insights."""
    if df is not None:
        # Ajouter une colonne pour l'année de publication
        if 'prism:coverDate' in df.columns:
            df['coverYear'] = pd.to_datetime(df['prism:coverDate'], errors='coerce').dt.year

        # Répartition des publications par année
        if 'coverYear' in df.columns:
            publications_per_year = df.groupby('coverYear')['dc:title'].count().reset_index(name='Publications')
            print("\nRépartition des publications par année :")
            print(publications_per_year)

    # Charger le DataFrame sauvegardé depuis le fichier CSV
    df = pd.read_csv('api_scopus_data.csv')

    # Analyser les données
    analyze_data(df)

Répartition des publications par année :
  coverYear  Publications
0          2024           25
```

Nous avons effectué une analyse des données de publications scientifiques en utilisant Pandas et Matplotlib. La fonction `analyze_data` prend un DataFrame `df` en entrée et ajoute une colonne pour l'année de publication si la colonne '`prism:coverDate`' est présente. Cette nouvelle colonne, '`coverYear`', est créée en convertissant les dates de publication en datetime et en extrayant l'année. Ensuite, la fonction calcule la répartition des publications par année en comptant le nombre de titres de publications pour chaque année, et affiche les résultats dans la console. Le DataFrame est chargé à partir d'un fichier CSV nommé '`api_scopus_data.csv`', puis la fonction `analyze_data` est appelée pour effectuer l'analyse.

• Corrélation entre Citations et Années de Publication :

```
def correlation_citations_annees(df):
    """Calcule la corrélation entre le nombre de citations et les années de publication."""
    if 'citedby-count' in df.columns and ('prism:coverDate' in df.columns or 'prism:coverDisplayDate' in df.columns):
        if 'prism:coverDate' in df.columns:
            df['coverYear'] = pd.to_datetime(df['prism:coverDate'], errors='coerce').dt.year
        elif 'prism:coverDisplayDate' in df.columns:
            df['coverYear'] = pd.to_datetime(df['prism:coverDisplayDate'], errors='coerce').dt.year

        correlation = df[['coverYear', 'citedby-count']].corr().iloc[0, 1]
        print(f"\nCorrélation entre le nombre de citations et les années de publication : {correlation:.2f}")
    else:
        print("Les colonnes nécessaires ('citedby-count' et 'prism:coverDate' ou 'prism:coverDisplayDate') ne sont pas présentes dans le DataFrame.")

# Appel de la fonction pour calculer la corrélation entre le nombre de citations et les années de publication
correlation_citations_annees(df)

Corrélation entre le nombre de citations et les années de publication : -0.24
```

Nous avons définir une fonction correlation_citations_annees qui calcule la corrélation entre le nombre de citations et les années de publication des articles dans un DataFrame Pandas. La fonction vérifie d'abord si les colonnes 'citedby-count' et soit 'prism:coverDate' ou 'prism:coverDisplayDate' sont présentes dans le DataFrame. Si 'prism:coverDate' est disponible, elle est convertie en format datetime et l'année est extraite pour créer une nouvelle colonne 'coverYear'. Si 'prism:coverDate' n'est pas disponible mais 'prism:coverDisplayDate' l'est, cette dernière est utilisée de la même manière. La corrélation entre les colonnes 'coverYear' et 'citedby-count' est ensuite calculée en utilisant la méthode corr() de Pandas, et le résultat est affiché. Si les colonnes nécessaires ne sont pas présentes, un message d'erreur est imprimé. Enfin, la fonction est appelée pour effectuer cette analyse sur le DataFrame chargé.

• Répartition des Publications en Accès Libre :

```
def publications_acces_libre(df):
    """Analyse la répartition des publications en accès libre."""
    if 'openaccessFlag' in df.columns:
        publications_open_access = df['openaccessFlag'].value_counts()
        print("\nRépartition des publications par statut d'accès libre :")
        print(publications_open_access)
    else:
        print("La colonne 'openaccessFlag' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications en accès libre
publications_acces_libre(df)

Répartition des publications par statut d'accès libre :
openaccessFlag
True      19
False       6
Name: count, dtype: int64
```

Nous avons définir une fonction publications_acces_libre qui analyse la répartition des publications en fonction de leur statut d'accès libre dans un DataFrame Pandas. La fonction vérifie d'abord si la colonne 'openaccessFlag' est présente dans le DataFrame. Si c'est le cas, elle utilise la méthode value_counts() pour compter le nombre de publications pour chaque valeur unique de la colonne 'openaccessFlag', qui indique le statut d'accès libre (par exemple, "Oui" ou "Non"). Ensuite, elle affiche les résultats de cette analyse, montrant la répartition des publications selon leur statut d'accès libre. Si la colonne requise n'est pas présente, un message d'erreur est imprimé pour informer que l'analyse ne peut pas être effectuée. Le DataFrame est supposé être préalablement chargé avec les données nécessaires avant d'appeler cette fonction pour analyser la répartition des publications en accès libre.

- Répartition des Publications par Source :

```
def publications_par_source(df):
    """Analyse la répartition des publications par source."""
    if 'prism:publicationName' in df.columns:
        publications_by_source = df['prism:publicationName'].value_counts().head(10)
        print("\nRépartition des publications par source (Top 10) :")
        print(publications_by_source)
    else:
        print("La colonne 'prism:publicationName' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par source
publications_par_source(df)

Répartition des publications par source (Top 10) :
prism:publicationName
Journal of Advanced Research in Applied Sciences and Engineering Technology      3
Dados                                         3
International Journal of Religion and Spirituality in Society                  2
Journal of Environmental Sciences (China)                                     1
Behavioral and Brain Sciences                                              1
Substance Abuse: Treatment, Prevention, and Policy                         1
Sports Medicine - Open                                                       1
Experimental Hematology and Oncology                                       1
Journal of Clinical and Translational Science                                1
Chinese Journal of Tissue Engineering Research                            1
```

Ce code définit une fonction `publications_par_source` qui analyse la répartition des publications par source dans un DataFrame Pandas. La fonction commence par vérifier si la colonne `'prism:publicationName'` est présente dans le DataFrame. Si c'est le cas, elle utilise la méthode `value_counts()` pour compter le nombre de publications par chaque nom de source et sélectionne les 10 premières sources avec le plus grand nombre de publications à l'aide de `head(10)`. Ces résultats sont ensuite affichés avec un titre indiquant "Répartition des publications par source (Top 10)". Si la colonne requise n'est pas présente, un message d'erreur est imprimé pour informer que la colonne `'prism:publicationName'` n'est pas présente dans le DataFrame. La fonction est ensuite appelée pour analyser et afficher la répartition des publications par source basée sur les données fournies dans le DataFrame `df`.

- Présence des Identifiants PubMed dans les Publications :

```
def publications_pubmed_id(df):
    """Analyse la présence des identifiants PubMed dans les publications."""
    if 'pubmed-id' in df.columns:
        publications_with_pubmed_id = df['pubmed-id'].notna().sum()
        total_publications = len(df)
        print(f"\nNombre de publications avec identifiant PubMed : {publications_with_pubmed_id} sur {total_publications} publications au total.")
    else:
        print("La colonne 'pubmed-id' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la présence des identifiants PubMed dans les publications
publications_pubmed_id(df)

Nombre de publications avec identifiant PubMed : 3 sur 25 publications au total.
```

Le code définit une fonction publications_pubmed_id qui analyse la présence des identifiants PubMed dans un DataFrame Pandas. La fonction vérifie d'abord si la colonne 'pubmed-id' est présente dans le DataFrame. Si c'est le cas, elle compte le nombre de publications pour lesquelles l'identifiant PubMed est disponible en utilisant la méthode notna() pour détecter les valeurs non nulles, puis en appliquant sum() pour compter ces occurrences. Elle calcule également le nombre total de publications dans le DataFrame. Ensuite, la fonction affiche le nombre de publications ayant un identifiant PubMed sur le nombre total de publications. Si la colonne 'pubmed-id' n'est pas présente, un message d'erreur est affiché indiquant que l'analyse ne peut pas être effectuée. Enfin, la fonction est appelée pour analyser la présence des identifiants PubMed dans le DataFrame chargé.

• Répartition des Publications par Type de Source :

```
def publications_par_type_source(df):
    """Analyse la répartition des publications par type de source (aggregation type)."""
    if 'prism:aggregationType' in df.columns:
        publications_by_aggregation_type = df['prism:aggregationType'].value_counts()
        print("\nRépartition des publications par type de source (Aggregation Type) :")
        print(publications_by_aggregation_type)
    else:
        print("La colonne 'prism:aggregationType' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par type de source
publications_par_type_source(df)

Répartition des publications par type de source (Aggregation Type) :
prism:aggregationType
Journal      24
Book         1
Name: count, dtype: int64
```

La fonction publications_par_type_source analyse la répartition des publications dans un DataFrame Pandas selon le type de source, identifié par la colonne 'prism:aggregationType'. Si cette colonne est présente dans le DataFrame (df), la fonction utilise la méthode value_counts() pour compter le nombre de publications pour chaque type de source. Les résultats sont ensuite affichés dans la console, indiquant le nombre de publications pour chaque type d'agrégation. Si la colonne requise n'est pas présente, un message d'erreur est imprimé, indiquant que 'prism:aggregationType' n'est pas disponible dans le DataFrame. Cette fonction permet ainsi d'obtenir un aperçu rapide de la répartition des publications en fonction de leur type de source, facilitant l'analyse exploratoire des données bibliographiques.

• Répartition des Publications par Sous-Type :

```
def publications_par_subtype(df):
    """Analyse la répartition des publications par type de sous-type."""
    if 'subtype' in df.columns:
        publications_by_subtype = df['subtype'].value_counts().head(10)
        print("\nRépartition des publications par type de sous-type (Top 10) :")
        print(publications_by_subtype)
    else:
        print("La colonne 'subtype' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par type de sous-type
publications_par_subtype(df)
```

```
Répartition des publications par type de sous-type (Top 10) :
subtype
ar    20
re     4
bk     1
Name: count, dtype: int64
```

La fonction `publications_par_subtype` qui analyse la répartition des publications par type de sous-type dans un DataFrame Pandas. La fonction commence par vérifier si la colonne 'subtype' est présente dans le DataFrame. Si c'est le cas, elle utilise la méthode `value_counts()` pour compter le nombre d'occurrences de chaque sous-type de publication et sélectionne les 10 premiers résultats les plus fréquents à l'aide de `head(10)`. Ces résultats sont ensuite affichés, montrant la répartition des publications par type de sous-type les plus courants. Si la colonne 'subtype' n'est pas présente, un message d'erreur est imprimé indiquant que l'analyse ne peut pas être effectuée. La fonction est appelée pour analyser la répartition des publications par type de sous-type sur le DataFrame chargé.

python

• Répartition des Publications par ISSN :

```
def publications_par_issn(df):
    """Analyse la répartition des publications par ISSN."""
    if 'prism:issn' in df.columns:
        publications_by_issn = df['prism:issn'].value_counts().head(10)
        print("\nRépartition des publications par ISSN (Top 10) :")
        print(publications_by_issn)
    else:
        print("La colonne 'prism:issn' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par ISSN
publications_par_issn(df)
```

```
Répartition des publications par ISSN (Top 10) :
prism:issn
02688921    3
0930343X    2
21947228    2
01790358    1
18632483    1
11102608    1
```

La fonction publications_par_issn qui analyse la répartition des publications par ISSN dans un DataFrame Pandas. La fonction commence par vérifier si la colonne 'prism:issn' est présente dans le DataFrame df. Si c'est le cas, elle utilise la méthode value_counts() pour compter le nombre d'occurrences de chaque ISSN et récupère les 10 premiers résultats les plus fréquents à l'aide de head(10). Ces résultats sont ensuite affichés à l'écran avec un message indiquant qu'il s'agit du top 10 des ISSN les plus fréquents dans le DataFrame. Si la colonne 'prism:issn' n'est pas présente, un message d'erreur est affiché pour informer que cette information n'est pas disponible dans le DataFrame actuel

• Répartition des Publications par Volume :

```
def publications_par_volume(df):
    """Analyse la répartition des publications par type de volume."""
    if 'prism:volume' in df.columns:
        publications_by_volume = df['prism:volume'].value_counts().head(10)
        print("\nRépartition des publications par type de volume (Top 10) :")
        print(publications_by_volume)
    else:
        print("La colonne 'prism:volume' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par type de volume
publications_par_volume(df)
```

```
Répartition des publications par type de volume (Top 10) :
prism:volume
68.0    3
19.0    2
43.0    2
27.0    1
10.0    1
13.0    1
8.0     1
28.0    1
77.0    1
14.0    1
```

La fonction publications_par_volume qui analyse la répartition des publications en fonction du type de volume dans un DataFrame Pandas. La fonction commence par vérifier si la colonne 'prism:volume' est présente dans le DataFrame. Si c'est le cas, elle utilise la méthode value_counts() pour compter le nombre d'occurrences de chaque type de volume et récupère les 10 premiers types avec le plus grand nombre de publications. Ces résultats sont ensuite affichés à l'écran avec un titre indiquant "Répartition des publications par type de volume (Top 10)". Si la colonne 'prism:volume' n'est pas trouvée dans le DataFrame, un message d'erreur est affiché pour informer l'utilisateur que cette colonne est absente. La fonction est appelée pour effectuer cette analyse sur le DataFrame df chargé à partir d'une source de données externe.

• Répartition des Publications par Type de Description de Sous-Type:

```
def publications_par_subtype_description(df):
    """Analyse la répartition des publications par type de description de sous-type."""
    if 'subtypeDescription' in df.columns:
        publications_by_subtype_desc = df['subtypeDescription'].value_counts().head(10)
        print("\nRépartition des publications par type de description de sous-type (Top 10) :")
        print(publications_by_subtype_desc)
    else:
        print("La colonne 'subtypeDescription' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par type de description de sous-type
publications_par_subtype_description(df)
```

```
Répartition des publications par type de description de sous-type (Top 10) :
subtypeDescription
Review      14
Article     11
...
```

La fonction `publications_par_subtype_description` analyse la répartition des publications en fonction de la description du sous-type dans un DataFrame Pandas. La fonction commence par vérifier si la colonne 'subtypeDescription' est présente dans le DataFrame. Si c'est le cas, elle utilise la méthode `value_counts()` pour compter le nombre de publications pour chaque type de description de sous-type, puis affiche les dix premiers résultats les plus fréquents. Ces résultats sont imprimés dans la console avec un titre indiquant qu'il s'agit des dix principaux types de description de sous-type pour les publications. Si la colonne 'subtypeDescription' n'est pas présente, un message d'avertissement est affiché. La fonction est ensuite appelée pour analyser la répartition des publications par type de description de sous-type dans le DataFrame chargé.

• Répartition des Publications par Numéro d'Article :

```
def publications_par_article_number(df):
    """Analyse la répartition des publications par numéro d'article."""
    if 'article-number' in df.columns:
        publications_by_article_number = df['article-number'].value_counts().head(10)
        print("\nRépartition des publications par numéro d'article (Top 10) :")
        print(publications_by_article_number)
    else:
        print("La colonne 'article-number' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par numéro d'article
publications_par_article_number(df)
```

```
Répartition des publications par numéro d'article (Top 10) :
article-number
e378      1
011005    1
e20220116  1
e20220091  1
e20220167  1
12         1
e42        1
e33        1
103587    1
e3         1
```

La fonction publications_par_article_number analyse la répartition des publications par numéro d'article dans un DataFrame Pandas. La fonction vérifie d'abord si la colonne 'article-number' est présente dans le DataFrame. Si c'est le cas, elle utilise la méthode value_counts() pour compter le nombre de publications pour chaque numéro d'article, puis sélectionne les 10 premiers résultats avec head(10). Ces résultats sont ensuite affichés dans la console, indiquant la répartition des publications par numéro d'article pour les articles les plus fréquents. Si la colonne 'article-number' n'est pas trouvée dans le DataFrame, un message d'erreur est affiché pour informer de l'absence de cette colonne. Le code montre comment utiliser Pandas pour effectuer une analyse simple mais informative de la distribution des publications en fonction de leur numéro d'article.

• Répartition des Publications par Type de Sous-catégorie :

```
def publications_par_sous_categorie(df):
    """Analyse la répartition des publications par type de sous-catégorie."""
    if 'subtypeDescription' in df.columns:
        publications_by_subtype = df['subtypeDescription'].value_counts().head(10)
        print("\nRépartition des publications par type de sous-catégorie (Top 10) :")
        print(publications_by_subtype)
    else:
        print("La colonne 'subtypeDescription' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par type de sous-catégorie
publications_par_sous_categorie(df)
```

```
Répartition des publications par type de sous-catégorie (Top 10) :
subtypeDescription
Article      20
Review        4
Book          1
```

La fonction publications_par_sous_categorie analyse la répartition des publications en fonction de leur sous-catégorie dans un DataFrame Pandas. La fonction commence par vérifier si la colonne 'subtypeDescription' est présente dans le DataFrame. Si c'est le cas, elle utilise la méthode value_counts() pour compter le nombre de publications pour chaque sous-catégorie et sélectionne les 10 premières sous-catégories avec le plus grand nombre de publications. Ces résultats sont ensuite affichés dans la console. Si la colonne requise n'est pas présente, un message d'erreur est affiché indiquant que la colonne 'subtypeDescription' n'est pas disponible. Enfin, la fonction est appelée pour effectuer cette analyse sur le DataFrame chargé.

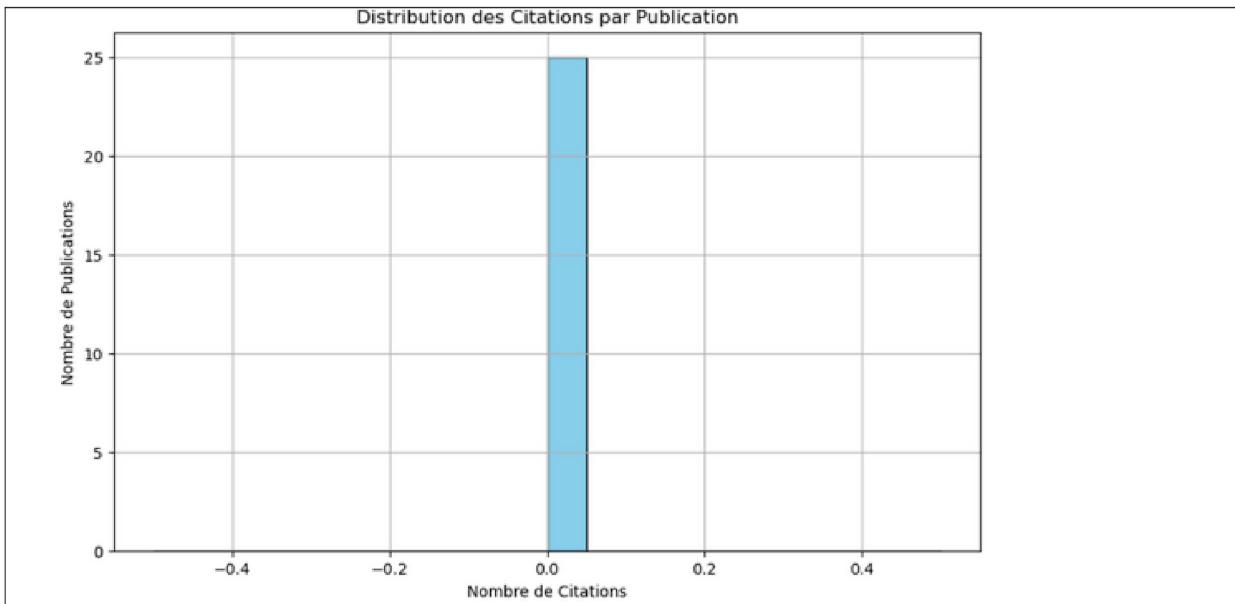
4. Visualisation des Données:

- Distribution des Citations par Publication :

```
import matplotlib.pyplot as plt

def citations_distribution(df):
    """Visualizes the distribution of citations."""
    if 'citedby-count' in df.columns:
        plt.figure(figsize=(10, 6))
        plt.hist(df['citedby-count'], bins=20, color='skyblue', edgecolor='black')
        plt.title('Distribution des Citations par Publication')
        plt.xlabel('Nombre de Citations')
        plt.ylabel('Nombre de Publications')
        plt.grid(True)
        plt.show()
    else:
        print("La colonne 'citedby-count' n'est pas présente dans le DataFrame.")

# Appel de la fonction
citations_distribution(df)
```



La fonction `plot_citations_distribution` crée un histogramme qui illustre comment les citations sont réparties parmi les publications. Elle filtre d'abord les données pour exclure les publications sans citations. Ensuite, elle génère un histogramme où chaque barre représente le nombre de publications en fonction du nombre de citations reçues, utilisant 20 bins pour regrouper les données. Les barres sont colorées en bleu ciel avec des bordures noires pour une meilleure visibilité. L'axe des x indique le nombre de citations par publication, tandis que l'axe des y indique le nombre correspondant de publications. Le titre 'Distribution des Citations par Publication' est ajouté pour identifier clairement le contenu du graphique. Si la colonne 'citedby-count' n'est pas présente dans les données, un message d'avertissement est affiché à la place.

- Distribution des Citations par année :

```

import pandas as pd
import matplotlib.pyplot as plt

# Charger le DataFrame sauvegardé depuis le fichier CSV
df = pd.read_csv('api_scopus_data.csv')

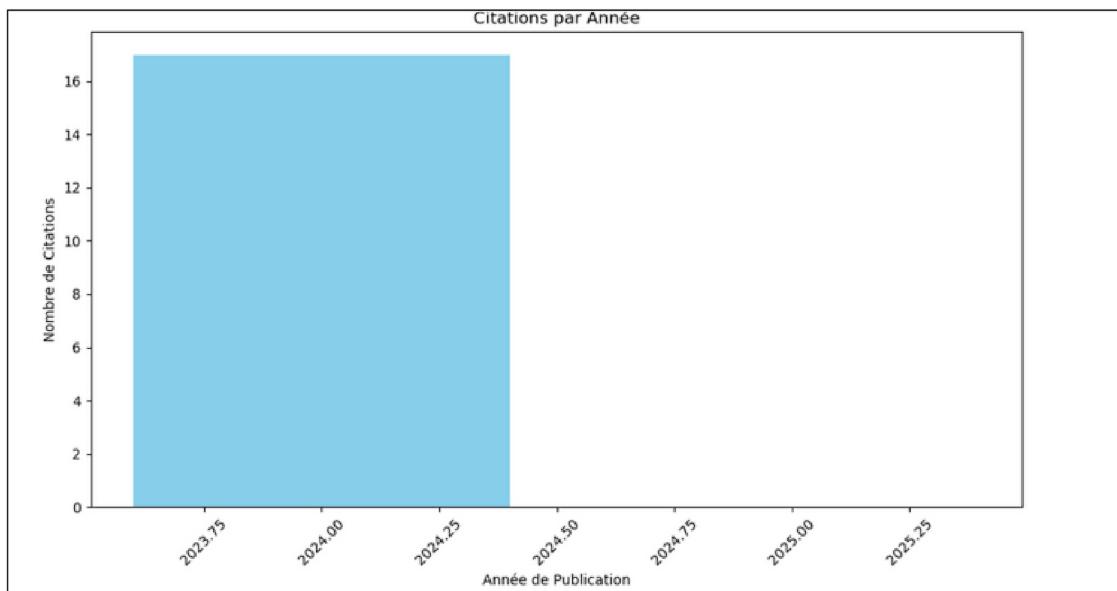
def citations_per_year(df):
    """Analyse les citations par année."""
    if 'citedby-count' in df.columns and 'prism:coverDate' in df.columns:
        # Convertir prism:coverdate en année
        df['coverYear'] = pd.to_datetime(df['prism:coverDate'], errors='coerce').dt.year

        citations_by_year = df.groupby('coverYear')['citedby-count'].sum()
        print("\nCitations par année :")
        print(citations_by_year)
        return citations_by_year # Retourne les citations par année
    else:
        print("Les colonnes nécessaires ('citedby-count' et 'prism:coverDate') ne sont pas présentes dans le DataFrame.")
        return None

# Appel de la fonction pour obtenir les citations par année
citations_by_year = citations_per_year(df)

# Vérification si les données ont été correctement chargées
if citations_by_year is not None:
    # Visualisation des citations par année
    plt.figure(figsize=(10, 6))
    plt.bar(citations_by_year.index, citations_by_year.values, color='skyblue')
    plt.xlabel('Année de Publication')
    plt.ylabel('Nombre de Citations')
    plt.title('Citations par Année')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
else:
    print("Impossible de visualiser les données car les citations par année n'ont pas été calculées correctement.")

```

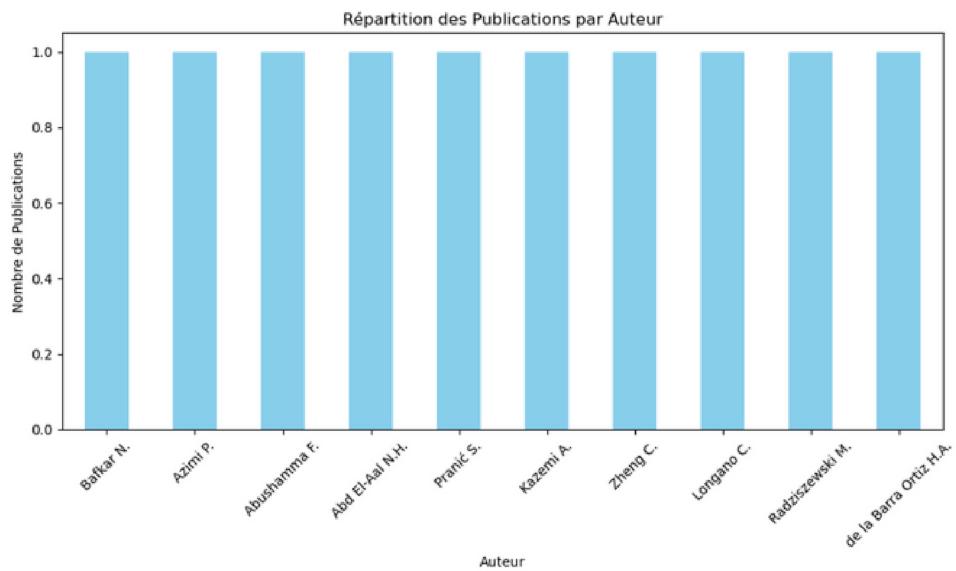


Cette visualisation présente le nombre de citations par année de publication des articles analysés dans le cadre du projet. Les données ont été agrégées et calculées à partir du DataFrame chargé à partir de la base de données Scopus. Chaque barre du graphique représente une année, avec l'axe vertical indiquant le nombre total de citations reçues par les publications au cours de cette année. Cette représentation graphique permet de mettre en évidence les tendances et les variations dans la popularité des articles au fil du temps, fournissant ainsi un aperçu précieux des impacts cumulatifs des recherches étudiées.

• Répartition des Publications par Auteur :

```
def publications_per_author(df):
    """Analyse la répartition des publications par auteur."""
    if 'dc:creator' in df.columns:
        publications_by_author = df['dc:creator'].value_counts()
        # Visualisation des publications par auteur (10 premiers auteurs)
        plt.figure(figsize=(10, 6))
        publications_by_author.head(10).plot(kind="bar", color="skyblue")
        plt.xlabel('Auteur')
        plt.ylabel('Nombre de Publications')
        plt.title('Répartition des Publications par Auteur')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()
    else:
        print("La colonne 'dc:creator' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour analyser la répartition des publications par auteur
publications_per_author(df)
```

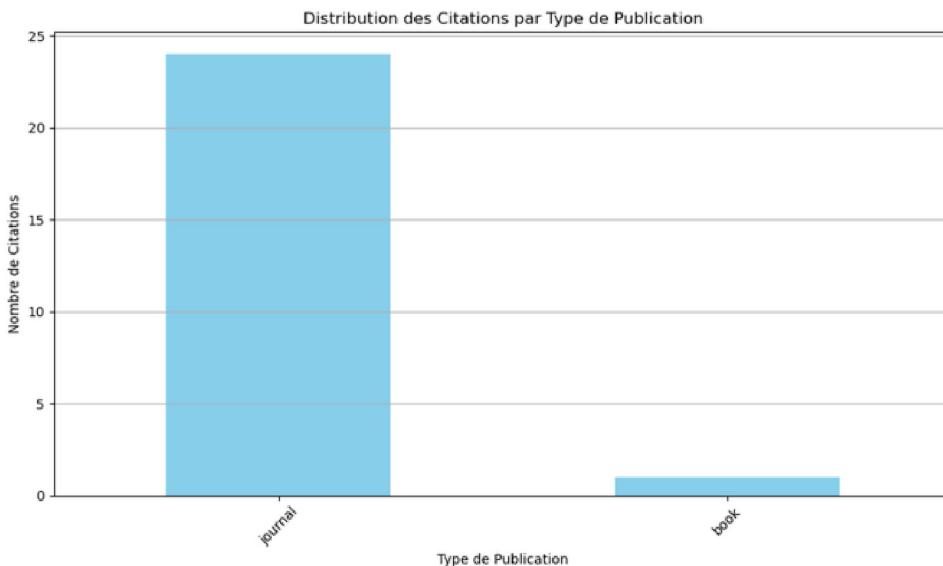


La fonction `publications_per_author` analyse la répartition des publications par auteur dans un DataFrame Pandas contenant des données de publications scientifiques. Elle vérifie d'abord si la colonne 'dc:creator' est présente dans le DataFrame. Si tel est le cas, elle utilise la méthode `value_counts()` pour compter le nombre de publications pour chaque auteur. Les résultats sont affichés dans la console, montrant les 10 premiers auteurs avec le plus grand nombre de publications. Ensuite, une visualisation sous forme de diagramme à barres est générée pour illustrer cette répartition pour les 10 premiers auteurs. Si la colonne requise n'est pas présente, un message d'erreur est affiché indiquant que la colonne 'dc:creator' n'est pas disponible. Cette analyse fournit un aperçu utile de la contribution des différents auteurs aux publications analysées.

• Distribution des Citations par Type de Publication :

```
def distribution_citations_par_type(df):
    """Visualise la distribution des citations par type de publication."""
    if 'citedby-count' in df.columns and 'prism:aggregationType' in df.columns:
        plt.figure(figsize=(10, 6))
        df_filtered = df[df['prism:aggregationType'].notna()]
        df_filtered['prism:aggregationType'] = df_filtered['prism:aggregationType'].str.lower()
        df_filtered['prism:aggregationType'].value_counts().plot(kind='bar', color='skyblue')
        plt.xlabel('Type de Publication')
        plt.ylabel('Nombre de Citations')
        plt.title('Distribution des Citations par Type de Publication')
        plt.grid(axis='y')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()
    else:
        print("Les colonnes nécessaires ('citedby-count' et 'prism:aggregationType') ne sont pas présentes dans le DataFrame.")

# Appel de la fonction pour visualiser la distribution des citations par type de publication
distribution_citations_par_type(df)
```



La fonction `distribution_citations_par_type` permet de visualiser la distribution des citations en fonction du type de publication dans un DataFrame Pandas. Elle commence par vérifier la présence des colonnes '`citedby-count`' (nombre de citations) et '`prism:aggregationType`' (type d'agrégation de la publication). Si ces colonnes sont disponibles, elle filtre le DataFrame pour exclure les lignes où le type d'agrégation n'est pas spécifié (NaN), convertit les valeurs de '`prism:aggregationType`' en minuscules pour uniformiser, puis trace un graphique en barres montrant le nombre de citations pour chaque type de publication. Les étiquettes des axes sont ajoutées pour indiquer le type de publication sur l'axe des x et le nombre de citations sur l'axe des y. Le titre du graphique est "Distribution des Citations par Type de Publication". Si les colonnes nécessaires ne sont pas présentes, un message est affiché indiquant que les données nécessaires ne sont pas disponibles pour effectuer cette analyse.

- Visualisation des Publications les Plus Citées :

```

import matplotlib.pyplot as plt

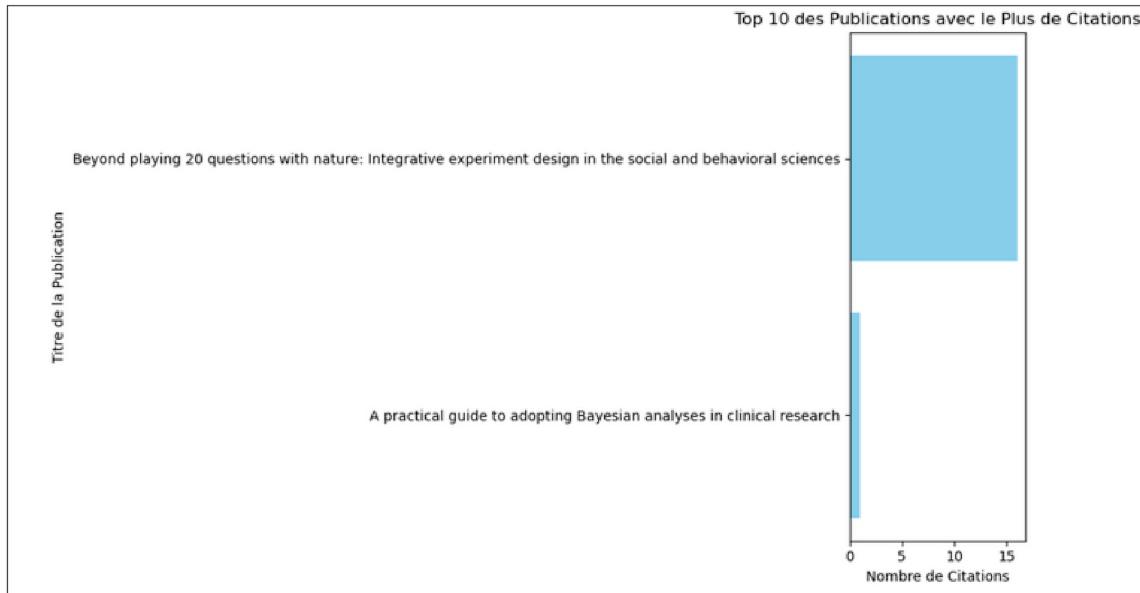
def plot_citations_per_publication(df):
    """Plot the number of citations per publication."""
    if 'citedby-count' in df.columns:
        # Filtrer les publications avec un nombre de citations non nul
        df_filtered = df[df['citedby-count'] > 0]

        # Tri des publications par nombre de citations (top 10)
        top_publications = df_filtered.nlargest(10, 'citedby-count')

        # Création du graphique à barres
        plt.figure(figsize=(10, 6))
        plt.barh(top_publications['dc:title'], top_publications['citedby-count'], color='skyblue')
        plt.xlabel('Nombre de Citations')
        plt.ylabel('Titre de la Publication')
        plt.title('Top 10 des Publications avec le Plus de Citations')
        plt.gca().invert_yaxis() # Inverser l'ordre des publications pour afficher du plus grand au plus petit
        plt.tight_layout()
        plt.show()
    else:
        print("La colonne 'citedby-count' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour visualiser le nombre de citations par publication
plot_citations_per_publication(df)

```



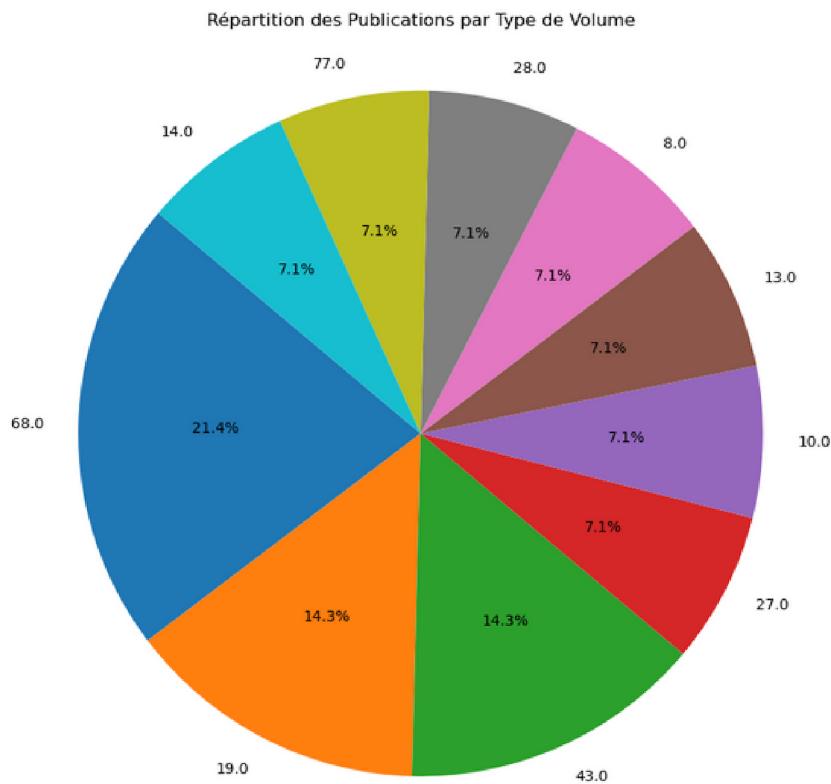
Cette fonction Python utilise Matplotlib pour générer un graphique à barres représentant les dix publications les plus citées dans un ensemble de données. Elle filtre d'abord les publications avec un nombre de citations supérieur à zéro, puis trie et sélectionne les dix publications ayant le plus grand nombre de citations. Le graphique résultant présente le titre de chaque publication sur l'axe vertical et le nombre de citations sur l'axe horizontal, avec une couleur de ciel pour une visualisation claire. Cette analyse est particulièrement utile pour identifier les publications les plus influentes dans un domaine donné, basée sur les données disponibles dans le DataFrame chargé.

- Répartition des Publications par Type de Volume :

```
def plot_publications_by_volume(df):
    """Plot the distribution of publications by volume type."""
    if 'prism:volume' in df.columns:
        # Compter le nombre de publications par type de volume (top 10)
        publications_by_volume = df['prism:volume'].value_counts().head(10)

        # Création du graphique à secteurs
        plt.figure(figsize=(8, 8))
        plt.pie(publications_by_volume.values, labels=publications_by_volume.index, autopct='%1.1f%%', startangle=140)
        plt.title('Répartition des Publications par Type de Volume')
        plt.axis('equal')
        plt.tight_layout()
        plt.show()
    else:
        print("La colonne 'prism:volume' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour visualiser la répartition des publications par type de volume
plot_publications_by_volume(df)
```



La fonction `plot_publications_by_volume` permet de visualiser la répartition des publications selon leur type de volume dans un ensemble de données de publications scientifiques. Elle analyse le DataFrame pour compter le nombre de publications pour chaque type de volume, en sélectionnant les 10 types les plus fréquents. Les résultats sont représentés sous forme de diagramme à secteurs, où chaque secteur correspond à un type de volume spécifique. Cette visualisation offre un aperçu clair de la distribution des publications par volume, facilitant ainsi l'identification des types de volumes prédominants dans les données analysées. Si la colonne 'prism:volume' n'est pas présente dans le DataFrame, un message indiquant cette absence est affiché pour informer l'utilisateur.

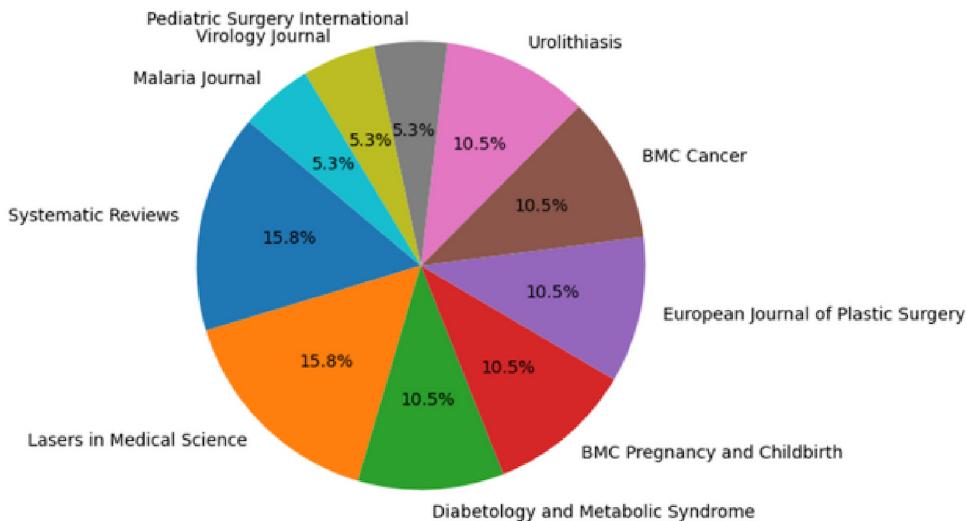
- Analyse de la Répartition des Publications par Source :

```
def plot_publications_by_source(df):
    """Plot the distribution of publications by source."""
    if 'prism:publicationName' in df.columns:
        # Compter le nombre de publications par source (top 10)
        publications_by_source = df['prism:publicationName'].value_counts().head(10)

        # Création du graphique à secteurs
        plt.figure(figsize=(8, 8))
        plt.pie(publications_by_source.values, labels=publications_by_source.index, autopct="%1.1f%%", startangle=140)
        plt.title("Répartition des Publications par Source")
        plt.axis('equal')
        plt.tight_layout()
        plt.show()
    else:
        print("La colonne 'prism:publicationName' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour visualiser la répartition des publications par source
plot_publications_by_source(df)
```

Répartition des Publications par Source



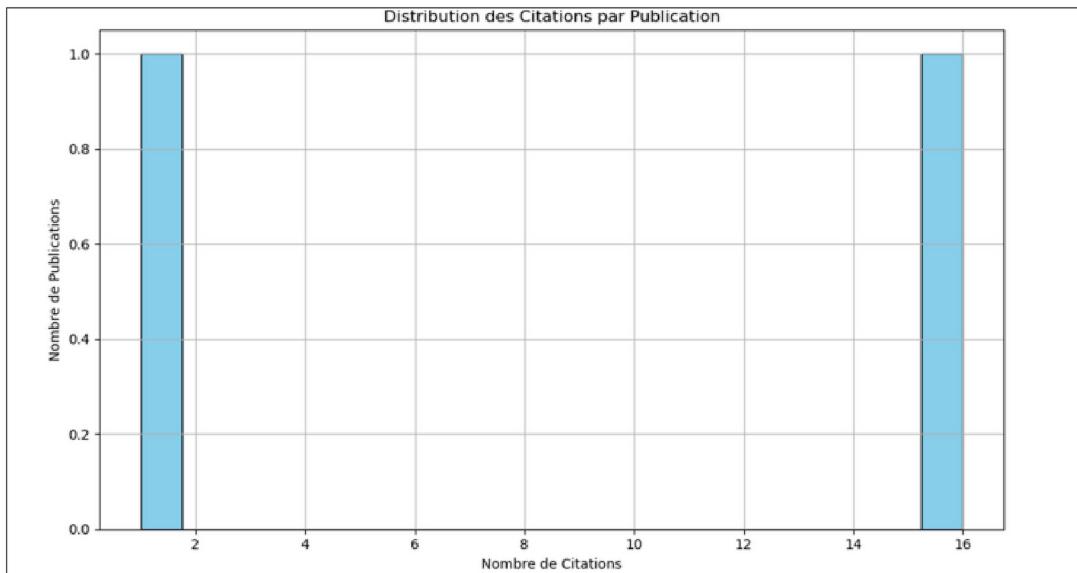
La fonction `plot_publications_by_source` permet de visualiser la répartition des publications scientifiques par source dans un ensemble de données donné. En vérifiant la présence de la colonne '`prism:publicationName`', elle compte le nombre de publications attribuées à chaque source et sélectionne les dix principales sources. Les résultats sont représentés graphiquement sous forme de diagramme circulaire (pie chart), montrant la proportion de publications pour chaque source en pourcentage. Ce graphique offre un aperçu visuel clair et concis de la contribution relative des différentes sources à l'ensemble des publications analysées. Si la colonne requise n'est pas disponible, un message est affiché indiquant que la colonne '`prism:publicationName`' n'est pas présente dans le DataFrame.

• Visualisation de la Distribution des Citations par Publication :

```
def plot_citations_distribution(df):
    """Plot the distribution of citations per publication."""
    if 'citedby-count' in df.columns:
        # Filtrer les publications avec un nombre de citations non nul
        df_filtered = df[df['citedby-count'] > 0]

        # Création de l'histogramme des citations
        plt.figure(figsize=(10, 6))
        plt.hist(df_filtered['citedby-count'], bins=20, color='skyblue', edgecolor='black')
        plt.xlabel('Nombre de Citations')
        plt.ylabel('Nombre de Publications')
        plt.title('Distribution des Citations par Publication')
        plt.grid(True)
        plt.tight_layout()
        plt.show()
    else:
        print("La colonne 'citedby-count' n'est pas présente dans le DataFrame.")

# Appel de la fonction pour visualiser la distribution des citations par publication
plot_citations_distribution(df)
```



La fonction `plot_citations_distribution` est utilisée pour générer un histogramme représentant la distribution du nombre de citations par publication dans un ensemble de données scientifiques. Le code commence par vérifier la présence de la colonne 'citedby-count' dans le DataFrame. Si cette colonne est présente, les publications avec un nombre de citations supérieur à zéro sont filtrées pour exclure celles sans citations. Ensuite, un histogramme est créé avec 20 bins, affichant le nombre de publications en fonction du nombre de citations reçues. Cette visualisation permet de comprendre la répartition des citations parmi les publications analysées, offrant ainsi un aperçu de leur impact et de leur popularité dans le domaine étudié. Si la colonne requise n'est pas trouvée, un message d'erreur est affiché indiquant que la colonne 'citedby-count' n'est pas disponible dans le DataFrame chargé.

5. RDF et SPARQL:

RDF (Resource Description Framework) : RDF est un cadre standardisé pour représenter des données sous forme de graphes, où chaque donnée est structurée sous forme de triplet sujet-prédicat-objet. Cela permet une représentation flexible et interopérable des données, facilitant leur intégration avec d'autres sources et leur manipulation.

SPARQL (SPARQL Protocol and RDF Query Language) : SPARQL est le langage de requête standard pour interroger des données RDF. Il offre la capacité d'effectuer des requêtes complexes et précises sur ces données structurées, permettant ainsi de récupérer des informations spécifiques, d'analyser des tendances, et de réaliser des calculs avancés.

- Installation de la bibliothèque rdflib avec Python

```
!pip install rdflib

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: rdflib in c:\users\intel\appdata\roaming\python\python311\site-packages (7.0.0)
Requirement already satisfied: isodate<0.7.0,>=0.6.0 in c:\users\intel\appdata\roaming\python\python311\site-packages (from rdflib) (0.6.1)
Requirement already satisfied: pyParsing<4,>=2.1.0 in c:\programdata\anaconda3\lib\site-packages (from rdflib) (3.0.9)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from isodate<0.7.0,>=0.6.0->rdflib) (1.16.0)
DEPRECATION: Loading egg at c:\programdata\anaconda3\lib\site-packages\vboxapi-1.0-py3.11.egg is deprecated. pip 24.3 will enforce this behaviour change. A possible replacement is to use pip for package installation.. Discussion can be found at https://github.com/pypa/pip/issues/12330
```

- Importation des Bibliothèques

```
import requests
import pandas as pd
from rdflib import Graph, Literal, RDF, URIRef, Namespace
from rdflib.namespace import DC
```

- Récupérer les Données depuis l'API Scopus

```
def fetch_scopus_data(api_key, query, count=25):
    """Fetch data from the Scopus API."""
    url = 'https://api.elsevier.com/content/search/scopus'
    params = {
        'apiKey': api_key,
        'query': query,
        'count': count
    }

    try:
        response = requests.get(url, params=params)
        response.raise_for_status()
        data = response.json()

        if 'search-results' in data and 'entry' in data['search-results']:
            print("La clé API est valide et fonctionne correctement.")
            print(f"Nombre de résultats obtenus : {len(data['search-results']['entry'])}")
            return data['search-results']['entry']
        else:
            print("La structure de la réponse JSON ne contient pas les clés attendues.")
            return None
    except requests.exceptions.HTTPError as http_err:
        print(f'Erreur HTTP (response.status_code): {response.reason}')
        print(response.text)
    except requests.exceptions.RequestException as req_err:
        print(f'Erreur de requête: {req_err}')
    except Exception as err:
        print(f'Erreur: {err}')
```

- Parser les Valeurs FreetoRead

```
def parse_freetoread(value):
    if isinstance(value, list):
        return ', '.join([item['$'] for item in value])
    return value
```

- Nettoyer et Sauvegarder les Données

```
def clean_and_save_data(entries, filename):
    if entries:
        df = pd.json_normalize(entries)

        if 'freetoread.value' in df.columns:
            df['freetoread.value'] = df['freetoread.value'].apply(parse_freetoread)

        pd.set_option('display.max_rows', None)
        pd.set_option('display.max_columns', None)
        pd.set_option('display.width', None)
        pd.set_option('display.max_colwidth', None)

        print(df)

        df.to_csv(filename, index=False)
        print(f"Les données ont été nettoyées et sauvegardées dans le fichier {filename}")
```

• Créer un RDF à partir d'un CSV

```

from rdflib import Graph, Namespace, Literal, URIRef
from rdflib.namespace import RDF, DC
import pandas as pd

def create_rdf_from_csv(csv_file, rdf_file):
    df = pd.read_csv(csv_file)
    g = Graph()

    SCOPUS = Namespace("http://example.org/scopus/")
    g.bind('scopus', SCOPUS)
    g.bind('dc', DC)

    for index, row in df.iterrows():
        publication = URIRef("http://example.org/scopus/publication/" + str(index))
        g.add((publication, RDF.type, SCOPUS.Publication))

        if 'dc:title' in row and pd.notna(row['dc:title']):
            g.add((publication, DC.title, Literal(row['dc:title'])))

        if 'citedby-count' in row and pd.notna(row['citedby-count']):
            g.add((publication, SCOPUS.citedByCount, Literal(row['citedby-count'])))

        if 'prism:publicationName' in row and pd.notna(row['prism:publicationName']):
            g.add((publication, SCOPUS.publicationName, Literal(row['prism:publicationName'])))

        if 'dc:creator' in row and pd.notna(row['dc:creator']):
            g.add((publication, DC.creator, Literal(row['dc:creator'])))

        if 'prism:coverDate' in row and pd.notna(row['prism:coverDate']):
            g.add((publication, DC.date, Literal(row['prism:coverDate'])))

        if 'freetoread.value' in row and pd.notna(row['freetoread.value']):
            g.add((publication, SCOPUS.freetoRead, Literal(row['freetoread.value'])))

        # Ajout des autres colonnes spécifiées
        if 'prism:eissn' in row and pd.notna(row['prism:eissn']):
            g.add((publication, SCOPUS.eissn, Literal(row['prism:eissn'])))

        if 'prism:volume' in row and pd.notna(row['prism:volume']):
            g.add((publication, SCOPUS.volume, Literal(row['prism:volume'])))

        if 'affiliation' in row and pd.notna(row['affiliation']):
            g.add((publication, SCOPUS.affiliation, Literal(row['affiliation'])))

        if 'prism:aggregationType' in row and pd.notna(row['prism:aggregationType']):
            g.add((publication, SCOPUS.aggregationType, Literal(row['prism:aggregationType'])))

        if 'subtype' in row and pd.notna(row['subtype']):
            g.add((publication, SCOPUS.subtype, Literal(row['subtype'])))

        if 'subtypeDescription' in row and pd.notna(row['subtypeDescription']):
            g.add((publication, SCOPUS.subtypeDescription, Literal(row['subtypeDescription'])))

        if 'article-number' in row and pd.notna(row['article-number']):
            g.add((publication, SCOPUS.articleNumber, Literal(row['article-number'])))

        if 'source-id' in row and pd.notna(row['source-id']):
            g.add((publication, SCOPUS.sourceId, Literal(row['source-id'])))

        if 'openaccess' in row and pd.notna(row['openaccess']):
            g.add((publication, SCOPUS.openAccess, Literal(row['openaccess'])))

        if 'openaccessFlag' in row and pd.notna(row['openaccessFlag']):
            g.add((publication, SCOPUS.openAccessFlag, Literal(row['openaccessFlag'])))

        if 'freetoread.value' in row and pd.notna(row['freetoread.value']):
            g.add((publication, SCOPUS.freetoRead, Literal(row['freetoread.value'])))

        if 'freetoreadLabel.value' in row and pd.notna(row['freetoreadLabel.value']):
            g.add((publication, SCOPUS.freetoReadLabel, Literal(row['freetoreadLabel.value'])))

        if 'prism:issn' in row and pd.notna(row['prism:issn']):
            g.add((publication, SCOPUS.issn, Literal(row['prism:issn'])))

        if 'pubmed-id' in row and pd.notna(row['pubmed-id']):
            g.add((publication, SCOPUS.pubmedId, Literal(row['pubmed-id'])))

        if 'coverYear' in row and pd.notna(row['coverYear']):
            g.add((publication, SCOPUS.coverYear, Literal(row['coverYear'])))

    g.serialize(rdf_file, format='turtle')

```

• Extraction et Visualisation des Informations de Publications Scopus

```
# Définition de votre code principal
api_key = '9ebde1fa88b0b7325c7d8054dd3e754'
query = 'KEY(scopus)'
filename = 'api_scopus_data.csv'
rdf_filename = 'scopus_data.ttl'

# Récupération et nettoyage des données
entries = fetch_scopus_data(api_key, query)
clean_and_save_data(entries, filename)

# Création du fichier RDF à partir du CSV
create_rdf_from_csv(filename, rdf_filename)
print(f"Les données RDF ont été créées à partir du fichier {filename}.")

def execute_sparql_query_and_style_results(rdf_filename, sparql_query):
    g = Graph()
    g.parse(rdf_filename, format='turtle')

    # Execute the SPARQL query
    results = g.query(sparql_query)

    # Styling the results
    print("Les noms de publication et dates : \n")
    for idx, row in enumerate(results):
        publication_name = row['publicationName']
        cover_date = row['coverDate']

        # Print each result with styling
        print(f"\t{idx + 1}. Publication Name: {publication_name}")
        print(f"\t\tDate: {cover_date}\n")

sparql_query_publications_info = """
PREFIX scopus: <http://example.org/scopus/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX prism: <http://prismstandard.org/namespaces/basic/2.0/>

SELECT ?publicationName ?coverDate
WHERE {
    ?publication a scopus:Publication ;
        dc:date ?coverDate ;
        scopus:publicationName ?publicationName .
}
LIMIT 5
"""
execute_sparql_query_and_style_results(rdf_filename, sparql_query_publications_info)
```

```
Les noms de publication et dates :
1. Publication Name: Systematic Reviews
Date: 2024-12-01

2. Publication Name: BMC Psychiatry
Date: 2024-12-01

3. Publication Name: BMC Pregnancy and Childbirth
Date: 2024-12-01

4. Publication Name: Thrombosis Journal
Date: 2024-12-01

5. Publication Name: Diabetology and Metabolic Syndrome
Date: 2024-12-01
```

La requête SPARQL suivante extrait les noms des publications et les dates de couverture des données RDF générées à partir d'un fichier CSV contenant des informations de Scopus. Le code parcourt les résultats et affiche chaque publication avec son nom et sa date.

• Extraction des Auteurs et Titres des Publications Scopus

```

def execute_sparql_query_authors_and_titles(rdf_filename, sparql_query):
    g = Graph()
    g.parse(rdf_filename, format='turtle')

    # Execute the SPARQL query
    results = g.query(sparql_query)

    # Store results in a list of tuples
    data = [(row['creator'], row['title']) for row in results]

    return data

# Votre requête SPARQL
sparql_query_authors_and_titles = """
PREFIX scopus: <http://example.org/scopuss/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?creator ?title
WHERE {
    ?publication a scopus:Publication .
    ?publication dc:creator ?creator .
    ?publication dc:title ?title .
}
LIMIT 5
"""

# Appel de la fonction pour exécuter la requête et obtenir les résultats sous forme de liste de tuples
results = execute_sparql_query_authors_and_titles(rdf_filename, sparql_query_authors_and_titles)

# Affichage des résultats sous forme de liste de tuples
print("\nCréateurs et titres des publications :")
for creator, title in results:
    print(f"Auteur: {creator}\nTitre: {title}\n")

```

Créateurs et titres des publications :

Auteur: Benavides-Gil G.
 Titre: Mindfulness-based interventions for improving mental health of frontline healthcare professionals during the COVID-19 pandemic: a systematic review and meta-analysis

Auteur: Bafkar N.
 Titre: Efficacy and safety of omega-3 fatty acids supplementation for anxiety symptoms: a systematic review and dose-response meta-analysis of randomized controlled trials

Auteur: Moradkhani A.
 Titre: Association of vitamin D receptor genetic polymorphisms with the risk of infertility: a systematic review and meta-analysis

Auteur: Maghsudlu M.
 Titre: Systematic review and meta-analysis of association between plasminogen activator inhibitor-1 4G/5G polymorphism and recurrent pregnancy loss: an update

Auteur: Miao Z.
 Titre: Impact of frailty on mortality, hospitalization, cardiovascular events, and complications in patients with diabetes mellitus: a systematic review and meta-analysis

Cette requête SPARQL extrait les auteurs (créateurs) et les titres des publications depuis un fichier RDF généré à partir des données de Scopus. Les résultats sont affichés sous forme de liste de tuples, chaque tuple contenant un auteur et le titre de sa publication.

• Analyse du Nombre de Publications par Année:

```
def execute_sparql_query_publications_by_year(rdf_file, sparql_query):
    # Fonction pour exécuter la requête SPARQL et afficher les résultats
    g = Graph()
    g.parse(rdf_file, format='turtle')

    qres = g.query(sparql_query)

    # Affichage des résultats sous forme de tableau
    print("\nNombre de publications par année :")
    print("{:<10} {:<10}".format("Year", "Count"))
    print("-" * 25)
    for row in qres:
        print("{:<10} {:<10}".format(row['year'], row['count']))

sparql_query_publications_by_year = """
PREFIX scopus: <http://example.org/scopus/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX prism: <http://prismstandard.org/namespaces/basic/2.0/>

SELECT ((?coverDate) AS ?year) (COUNT(?publication) AS ?count)
WHERE {
    ?publication a scopus:Publication ;
        dc:date ?coverDate .
}
GROUP BY ?year
"""
# Appel de la fonction avec la nouvelle requête et le nom de fonction modifié
execute_sparql_query_publications_by_year(rdf_filename, sparql_query_publications_by_year)
```

```
Nombre de publications par année :
Year      Count
-----
2024-12-01  25
```

Cette requête SPARQL compte le nombre de publications par année à partir d'un fichier RDF généré à partir des données de Scopus. Les résultats sont affichés sous forme de tableau, chaque ligne représentant une année et le nombre de publications correspondantes

- Affichage des DOI et Noms de Publication:

```

def execute_sparql_query_doi_and_publication_name(rdf_file, sparql_query):
    # Fonction pour exécuter la requête SPARQL et afficher les résultats
    g = Graph()
    g.parse(rdf_file, format='turtle')

    qres = g.query(sparql_query)

    # Affichage des résultats sous forme de tableau
    print("\nAffichage des DOI et noms de publication :")
    print("{:<30} {:<70}\n".format("DOI", "Publication Name"))
    print("-" * 100)
    for row in qres:
        print("{:<30} {:<70}\n".format(row['doi'], row['publicationName']))

sparql_query_doi_and_publication_name = """
PREFIX scopus: <http://example.org/scopus/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX prism: <http://prismstandard.org/namespaces/basic/2.0/>

SELECT ?doi ?publicationName
WHERE {
    ?publication a scopus:Publication ;
        scopus:doi ?doi ;
        scopus:publicationName ?publicationName .
}
LIMIT 6
"""
# Appel de la fonction avec la nouvelle requête et le nom de fonction modifié
execute_sparql_query_doi_and_publication_name(rdf_filename, sparql_query_doi_and_publication_name)

```

```

Affichage des DOI et noms de publication :
DOI                  Publication Name
-----
10.1186/s13643-024-02574-5  Systematic Reviews
10.1186/s12888-024-05881-2  BMC Psychiatry
10.1186/s12884-024-06590-0  BMC Pregnancy and Childbirth
10.1186/s12959-024-08612-9  Thrombosis Journal
10.1186/s13098-024-01352-6  Diabetology and Metabolic Syndrome
10.1007/s00238-024-02190-5  European Journal of Plastic Surgery

```

Cette requête SPARQL extrait les DOI et les noms de publication à partir d'un fichier RDF généré à partir des données de Scopus. Les résultats sont affichés sous forme de tableau, chaque ligne présentant un DOI et le nom de la publication correspondante.

• Affichage des Volumes des Publications

```
def execute_sparql_query_volume(rdf_file, sparql_query):
    # Fonction pour exécuter la requête SPARQL et afficher les résultats
    g = Graph()
    g.parse(rdf_file, format='turtle')

    qres = g.query(sparql_query)

    # Affichage des résultats sous forme de tableau
    print("\nAffichage des volumes des publications :")
    print("{:<10} {:<30}\n".format("Publication", "Volume"))
    print("-" * 50)
    for row in qres:
        print("{:<10} {:<30}\n".format(row['publication'], row['volume']))

sparql_query_volume = """
PREFIX scopus: <http://example.org/scopus/>
PREFIX prism: <http://prismstandard.org/namespaces/basic/2.0/>

SELECT ?publication ?volume
WHERE {
    ?publication a scopus:Publication ;
        scopus:volume ?volume .
}
LIMIT 10
"""

# Appel de la fonction avec la requête pour les volumes des publications
execute_sparql_query_volume(rdf_filename, sparql_query_volume)
```

```
Affichage des volumes des publications :
Publication Volume
-----
http://example.org/scopus/publication/0 13
http://example.org/scopus/publication/1 24
http://example.org/scopus/publication/10 22
http://example.org/scopus/publication/11 22
http://example.org/scopus/publication/12 16
http://example.org/scopus/publication/13 47
http://example.org/scopus/publication/14 24
http://example.org/scopus/publication/15 13
http://example.org/scopus/publication/16 24
http://example.org/scopus/publication/17 39
```

Cette requête SPARQL extrait les volumes des publications à partir d'un fichier RDF généré à partir des données de Scopus. Les résultats sont affichés sous forme de tableau, chaque ligne présentant une publication et le volume correspondant.

- Affichage des Articles et de leurs Volumes:

```

from rdflib import Graph, Namespace
def execute_sparql_query_article_volume(rdf_file, sparql_query):
    # Chargement du fichier RDF
    g = Graph()
    g.parse(rdf_file, format="turtle")

    # Exécution de la requête SPARQL
    qres = g.query(sparql_query)

    # Affichage des résultats
    print("\nPublications de type 'Article' avec leur volume :")
    print("-" * 50)
    for row in qres:
        print(f"Titre de la publication : {row['title']}")
        print(f"Volume : {row['volume']}")
        print("-" * 50)

    # Requête SPARQL pour récupérer les articles avec leur volume
    sparql_query_article_volume = """
PREFIX scopus: <http://example.org/scopus/>
PREFIX prism: <http://prismstandard.org/namespaces/basic/2.0/>

SELECT ?title (GROUP_CONCAT(?volume; separator=", ") AS ?volume)
WHERE {
    ?publication a scopus:Publication ;
        scopus:subtypeDescription "Article" ;
        scopus:volume ?volume ;
        dc:title ?title .
}
GROUP BY ?title
LIMIT 10
"""

    # Appel de la fonction avec la requête pour les articles et leur volume
    execute_sparql_query_article_volume(rdf_filename, sparql_query_article_volume)

```

```

Publications de type 'Article' avec leur volume :
-----
Titre de la publication : Mindfulness-based interventions for improving mental health of frontline healthcare professionals during the COVID-19 pandemic
Volume : 13
-----
Titre de la publication : Efficacy and safety of omega-3 fatty acids supplementation for anxiety symptoms: a systematic review and dose-response meta-analysis of randomized controlled trials
Volume : 24
-----
Titre de la publication : Association of vitamin D receptor genetic polymorphisms with the risk of infertility: a systematic review and meta-analysis
Volume : 24
-----
Titre de la publication : Impact of frailty on mortality, hospitalization, cardiovascular events, and complications in patients with diabetes mellitus: a systematic review and meta-analysis
Volume : 16
-----
Titre de la publication : Association of prothrombin time, thrombin time and activated partial thromboplastin time levels with preeclampsia: a systematic review and meta-analysis
Volume : 24
-----
Titre de la publication : Protocol for a systematic review and meta-analysis on Janus kinase inhibitors in the management of vitiligo
Volume : 13
-----
Titre de la publication : Analyzing global research trends and focal points in the utilization of laser techniques for the treatment of urolithiasis from 1978 to 2022: visualization and bibliometric analysis
Volume : 52
-----
Titre de la publication : Barriers and facilitators to implementing workplace interventions to promote mental health: qualitative evidence synthesis
Volume : 13
-----
Titre de la publication : The impact of immunosuppression on the mortality and hospitalization of Monkeypox: a systematic review and meta-analysis of the 2022 outbreak
Volume : 21
-----
Titre de la publication : 75 years' journey of malaria publications in English: what and where?
Volume : 23
-----
```

Cette requête SPARQL extrait les titres des publications de type "Article" ainsi que leurs volumes à partir d'un fichier RDF généré à partir des données de Scopus. Les résultats sont affichés sous forme de liste, chaque article étant accompagné de son volume.

Conclusion

En conclusion, ce projet a permis d'explorer et d'appliquer de manière pratique les compétences en manipulation de données, en analyse et en visualisation, en utilisant des outils et des bibliothèques Python tels que NumPy, Pandas, et Matplotlib. Grâce à l'API Scopus, nous avons pu automatiser la récupération des publications scientifiques et de leurs métadonnées, offrant ainsi un aperçu précieux des tendances et des distributions au sein des données récupérées.

Les étapes détaillées dans ce rapport montrent non seulement l'efficacité de l'utilisation des APIs pour l'extraction de données réelles, mais aussi l'importance de l'analyse et de la visualisation pour une meilleure compréhension des informations extraites. Les résultats obtenus confirment l'utilité de ces techniques pour l'analyse des publications scientifiques, et les conclusions tirées de cette analyse soulignent les tendances significatives et les observations pertinentes dans les domaines scientifiques explorés. Ce projet illustre ainsi la synergie entre les compétences en programmation, en analyse de données, et en visualisation pour aborder des questions complexes dans le domaine de la recherche scientifique.