

Rapport du projet python

Réalisé par : MOURID Fatine & SAADANE Manel

Introduction générale	2
I. Spécifications du projet	3
II. Analyse du projet	4
II.1 Environnement de travail	4
II.2 Données utilisées	4
II.3 Organisation du programme et classes utilisées.....	5
III. Conception du projet	6
III.1 Organisation du travail en binôme	6
III.2 Algorithmes et principes de fonctionnement	7
III.3 Problèmes rencontrés et solutions apportées	9
III.4 Exemple d'utilisation du programme.....	9
IV. Validation et tests	10
IV.1 Tests unitaires	10
IV.2 Tests globaux.....	11
V. Maintenance et perspectives	12
Conclusion.....	13

Introduction générale

Ce rapport présente un travail réalisé dans le cadre du projet de **programmation en Python**, réalisé en binôme par **Manel Saadane** et **Fatine Mourid**. L'objectif principal de ce projet est de mettre en place un **moteur de recherche d'information simple**, permettant d'analyser et de rechercher des informations dans des données textuelles.

Le projet s'appuie sur le **thème de la photographie**, choisi comme sujet principal d'étude. Les données utilisées sont des textes liés à ce thème, provenant de différentes sources telles que **Reddit**, **arXiv**, ainsi que des fichiers au format **CSV**. Ces données permettent de constituer un corpus représentatif autour de la photographie.

Dans ce rapport, la **première partie** présente les **spécifications du projet**, c'est-à-dire les objectifs et les fonctionnalités attendues. La **deuxième partie** est consacrée à **l'analyse**, où sont expliqués les choix concernant l'environnement de travail, les données utilisées et l'organisation générale du programme.

La **troisième partie** décrit la **conception** du projet. Elle explique comment le programme a été structuré, quels algorithmes ont été utilisés et quelles difficultés ont été rencontrées au cours du développement.

Ensuite, la partie **validation et tests** présente les vérifications réalisées pour s'assurer du bon fonctionnement du programme.

Enfin, la dernière partie aborde la **maintenance et les perspectives**, en proposant des améliorations possibles et des évolutions futures du projet.

I. Spécifications du projet

L'objectif de ce projet est de développer un **moteur de recherche d'information simple** en utilisant le langage **Python**. Ce moteur permet à un utilisateur d'effectuer des recherches à partir de **mots-clés** dans un ensemble de documents textuels.

Le programme doit être capable de **collecter, stocker, structurer, analyser et interroger** des données textuelles liées à un thème choisi. Dans le cadre de ce projet, le thème retenu est **la photographie**. Les données sont récupérées depuis des sources externes telles que **Reddit** et **arXiv**, puis sauvegardées dans des fichiers afin de pouvoir être réutilisées lors des exécutions suivantes.

Le projet doit permettre, étape par étape :

- La **récupération automatique** de données textuelles à partir de différentes sources (API Reddit et arXiv)
- La **sauvegarde et le rechargement** des données dans des fichiers au format **CSV**, afin d'éviter d'interroger les APIs à chaque exécution
- L'organisation des documents et de leurs métadonnées à l'aide de **classes orientées objet**
- La gestion de différents types de documents grâce à l'**héritage et au polymorphisme**, ainsi que l'utilisation de **patrons de conception**
- L'**analyse du contenu textuel**, incluant le nettoyage des textes, le découpage en mots, la recherche par expressions régulières et le calcul de statistiques textuelles
- La mise en place d'un **moteur de recherche**, basé sur des représentations vectorielles des documents (TF et TF-IDF) et sur une mesure de similarité
- L'intégration du moteur de recherche à partir d'un **jeu de données au format CSV**, afin de tester le moteur sur un corpus différent
- La création d'une **interface utilisateur** permettant d'interagir avec le moteur de recherche
- La réalisation d'un **mini-projet** proposant des fonctionnalités supplémentaires, telles que l'analyse avancée du corpus et la comparaison entre différentes sources de documents

Ces différentes fonctionnalités correspondent aux **étapes successives du projet**, réalisées progressivement au cours des **travaux dirigés TD3 à TD10**, depuis l'acquisition des données jusqu'au mini-projet final.

Enfin, le programme doit permettre une **interaction avec l'utilisateur**, notamment pour saisir des requêtes et afficher les résultats de recherche.

II. Analyse du projet

Cette partie présente **comment les spécifications du projet** ont été prises en compte et comment leur réalisation a été préparée. Elle décrit l'environnement de travail utilisé, les données manipulées et l'organisation générale du programme à travers les classes mises en place

II.1 Environnement de travail

Le projet a été développé en utilisant le langage **Python**.

L'environnement de développement principal utilisé est **Visual Studio Code (VS Code)**, choisi pour sa légèreté, sa compatibilité avec Python et ses outils facilitant l'écriture et l'organisation du code.

Les programmes ont été exécutés à l'aide du **terminal de l'ordinateur**, ce qui a permis de lancer les scripts Python, d'installer les bibliothèques nécessaires via pip et de tester le bon fonctionnement du programme.

Le projet a également été versionné à l'aide de **Git** et hébergé sur **GitHub**, afin de :

- Sauvegarder le travail,
- Suivre l'évolution du projet,

Pour certaines étapes (TD8 à TD10), un environnement **Anaconda** et des **notebooks Jupyter** ont été utilisés afin de faciliter les tests et l'exploration des données. Par la suite, le code a été réorganisé en scripts Python pour constituer la version finale du projet.

II.2 Données utilisées

Les données utilisées dans le projet sont des **données** liées au thème de la **photographie**, conformément aux spécifications.

Elles proviennent de plusieurs sources :

- **Reddit** : récupération de publications textuelles via l'API Reddit, à partir de mots-clés en lien avec le thème choisi ;
- **arXiv** : récupération de résumés d'articles scientifiques via l'API arXiv ;
- **fichiers CSV** : stockage des textes et de leurs métadonnées (auteur, source, date, etc.), afin de pouvoir recharger les données sans refaire les appels aux APIs.

Ces données sont ensuite regroupées pour former un **corpus de documents**, qui sert de base aux différentes analyses et au moteur de recherche.

II.3 Organisation du programme et classes utilisées

Afin de structurer le programme, une **approche orientée objet** a été adoptée.

Les principales classes utilisées sont :

- **Document** : classe représentant un document textuel général
- **RedditDocument** : sous-classe de Document, spécifique aux documents provenant de Reddit
- **ArxivDocument** : sous-classe de Document, spécifique aux documents provenant d'arXiv
- **Author** : classe représentant un auteur et les documents associés
- **Corpus** : classe centrale regroupant l'ensemble des documents et des auteurs
- **SearchEngine** : classe chargée de la recherche d'information à partir du corpus.
- **DocumentFactory** : classe de type *Factory* permettant de créer automatiquement le bon type de document (Reddit, arXiv ou générique) à partir des données chargées.

Les classes RedditDocument et ArxivDocument héritent de la classe Document. Cette relation d'héritage permet de partager des attributs communs tout en gérant des informations spécifiques à chaque source.

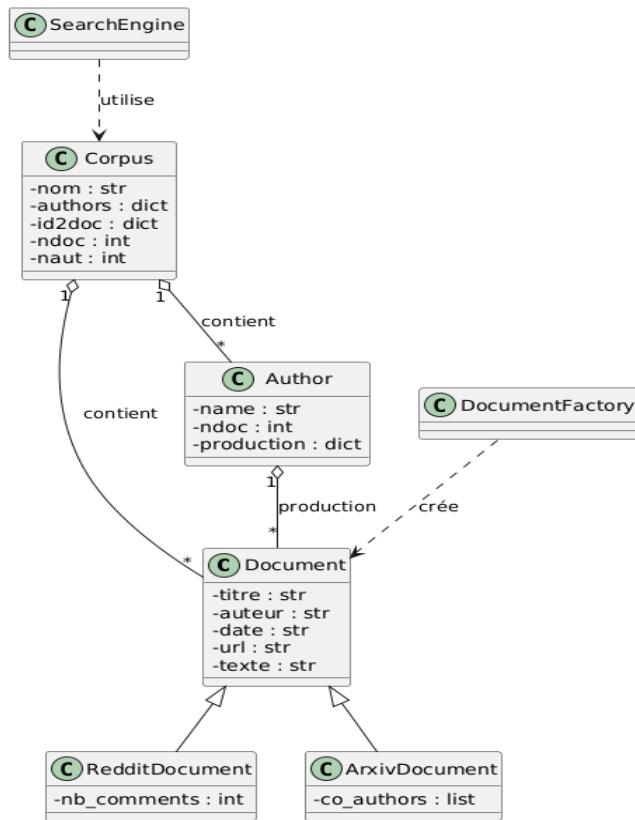
La classe Corpus utilise les classes Document et Author pour organiser les données, effectuer des analyses et fournir les informations nécessaires au moteur de recherche.

La classe SearchEngine utilise le corpus afin de calculer les scores de pertinence et de retourner les résultats correspondant à une requête utilisateur.

Cette organisation permet une **séparation claire des responsabilités**, une meilleure lisibilité du code et une évolution plus simple du programme.

Le diagramme de classes ci-dessous illustre l'organisation orientée objet du programme et les relations entre les principales classes utilisées.

Figure 1. Diagramme de classes UML simplifié



III. Conception du projet

Cette partie présente la manière dont l'analyse du projet a été **mise en œuvre concrètement**. Elle décrit l'organisation du travail en binôme, les principaux algorithmes utilisés, les difficultés rencontrées au cours du développement ainsi qu'un exemple d'utilisation du programme, du début à la fin.

III.1 Organisation du travail en binôme

Le travail a été mené de manière **collaborative**, principalement durant les séances de **travaux dirigés**, sur un **même poste de travail** (**ordinateur de Manel**).

Au début du projet, certaines difficultés ont été rencontrées pour bien comprendre l'objectif global du sujet et la progression attendue entre les différents TD. Pour cette raison, plusieurs travaux dirigés ont parfois été abordés **en parallèle**, afin de mieux relier les différentes étapes du projet et d'assurer une cohérence dans le développement.

Nous avons travaillé ensemble sur l'ensemble des TD, en alternant les phases de réflexion, de schématisation et d'implémentation. En dehors des séances de TD, le travail a également été poursuivi lors de rencontres régulières afin d'avancer sur certaines parties du projet, relire le code et préparer les livrables.

Un outil d'assistance tel que **ChatGPT** a également été utilisé comme **support**, notamment pour poser des questions ponctuelles sur la syntaxe Python, la structuration du code ou certains aspects liés à l'interface. Cet outil a été utilisé comme une aide à la compréhension et à la résolution de problèmes techniques, sans se substituer au travail personnel.

III.2 Algorithmes et principes de fonctionnement

Le projet repose sur une **suite d'algorithmes simples**, mis en place progressivement au fil des différents travaux dirigés, depuis la récupération des données jusqu'à la recherche d'information et l'exploration des corpus.

Dans un premier temps (**TD3**), des algorithmes de **récupération de données** sont utilisés pour collecter des textes à partir de sources externes telles que **Reddit** et **arXiv**. Les données récupérées sont ensuite **structurées** et **sauvegardées** dans des fichiers CSV afin de pouvoir être réutilisées sans interroger à nouveau les APIs.

Ensuite (**TD4 et TD5**), les documents sont organisés à l'aide de la **programmation orientée objet**. Des algorithmes simples permettent d'ajouter les documents au corpus, d'associer chaque document à son auteur et de gérer différents types de documents grâce à l'**héritage** et au **polymorphisme**.

Dans la phase d'analyse du texte (**TD6**), les documents textuels sont **nettoyés**, **découpés en mots** et analysés afin de construire un vocabulaire exploitable. Des algorithmes de recherche par motifs (expressions régulières) et de calcul de **statistiques textuelles**, telles que la fréquence des mots, sont également mis en place.

Pour la recherche d'information (**TD7**), chaque document est représenté sous forme de **vecteur** :

- une représentation **TF (Term Frequency)**, qui mesure la fréquence d'un mot dans un document ;
- une représentation **TF-IDF**, qui pondère les mots en fonction de leur importance dans l'ensemble du corpus.

Lorsqu'un utilisateur saisit une requête, celle-ci est traitée de la même manière que les documents. Une **mesure de similarité** est ensuite utilisée pour comparer la requête aux documents du corpus, ce qui permet de classer les documents par ordre de pertinence et de retourner les résultats les plus proches.

Enfin, les **TD8, TD9 et TD10** ont été initialement développés dans un environnement **Anaconda**, en utilisant des **notebooks Jupyter (.ipynb)**. Ce choix a permis de faciliter l'exploration

des données, les tests progressifs et la mise au point des différentes fonctionnalités, en particulier lors de la phase de prototypage.

Dans une étape suivante, les notebooks Jupyter ont été **exportés en scripts Python (.py)**. Le code obtenu a ensuite été **réorganisé et structuré** afin de s'intégrer correctement au reste du projet. Une **interface utilisateur** a alors été développée en Python à l'aide de la bibliothèque **Streamlit**.

Cette interface Streamlit reprend la **même logique de fonctionnement** que celle implémentée dans les notebooks Jupyter, tout en offrant une **exécution standardisée et interactive via un navigateur web**. Cette transition permet de conserver le travail réalisé durant les phases de développement tout en proposant une version finale du projet plus **modulaire** et plus **facilement exploitable**.

Les fichiers **.ipynb** sont ainsi conservés comme **support de développement et de prototypage**, tandis que les fichiers **.py** et l'interface Streamlit constituent la **version finale du projet**.

Le fichier **app.py** a été identifié comme le **fichier le plus complexe du projet**, car il centralise l'ensemble des fonctionnalités développées dans les TD précédents et gère les interactions avec l'utilisateur. Il assure la coordination entre le chargement des données, l'exécution du moteur de recherche, l'affichage des résultats et l'exploration du corpus.

L'interface suit une logique en plusieurs étapes :

- **Chargement du dataset** : l'utilisateur choisit le fichier CSV (par exemple discours_US.csv) ou un corpus déjà construit
- **Construction ou chargement du moteur** : l'application initialise le moteur de recherche à partir des données, en créant le corpus et en préparant les structures nécessaires au calcul des scores
- **Saisie de la requête** : l'utilisateur saisit des mots-clés et choisit le nombre de résultats à afficher (Top-K)
- **Recherche et affichage des résultats** : l'application calcule la pertinence des documents (TF / TF-IDF et mesure de similarité) puis affiche les résultats triés, accompagnés des informations associées
- **Exploration du corpus** : l'interface permet également de consulter certaines statistiques du corpus et de comparer des sous-ensembles de documents dans le cadre du mini-projet (**TD9-TD10**)

Cette organisation permet une **séparation claire** entre **le cœur du moteur** (modules de traitement et de recherche) et **l'interface utilisateur**, facilitant ainsi l'exécution, la compréhension et la réutilisation du projet.

Les détails concernant la **structure des fichiers**, les **prerequisites** et les **modalités d'exécution** sont précisés dans le fichier **README.txt** joint au projet.

III.3 Problèmes rencontrés et solutions apportées

Plusieurs difficultés ont été rencontrées au cours du développement du projet.

L'une des principales difficultés a été la **compréhension globale du sujet**, notamment la manière dont les différents TD s'articulaient entre eux. Cette difficulté a été progressivement surmontée en retravaillant certaines parties du projet, en revenant sur les TD précédents et en réorganisant le code.

D'autres problèmes ont également été rencontrés, tels que :

- La structuration du code en plusieurs fichiers et classes
- La gestion de différents types de documents (Reddit et arXiv)
- La sauvegarde et le rechargement des données à partir de fichiers CSV
- La transition entre des notebooks Jupyter et des scripts Python exécutables.

Ces difficultés ont été résolues grâce à des tests réguliers, des échanges au sein du binôme et une réorganisation progressive du code afin de le rendre plus lisible et plus modulaire.

III.4 Exemple d'utilisation du programme

Un exemple d'utilisation typique du programme est le suivant. L'utilisateur lance l'exécution avec la commande :

- `python main.py`

Au démarrage, le programme vérifie si un fichier de corpus existe déjà (par exemple `data/corpus.csv`).

- **S'il existe**, le corpus est **chargé automatiquement** depuis le fichier CSV, ce qui évite de refaire des appels aux APIs.
- **Sinon**, le programme interroge automatiquement les APIs **Reddit** et **arXiv** à partir d'un thème défini (par défaut **photography**) et récupère un nombre limité de documents, puis **sauvegarde** le résultat dans un fichier CSV pour les exécutions suivantes.

Après le chargement ou la récupération, le programme réalise des traitements et affichages intermédiaires (statistiques simples, filtrage des documents trop courts, recherche par motif et concordancier). Ensuite, le moteur de recherche est lancé : le programme affiche une demande de saisie, et l'utilisateur tape une requête, par exemple :

- Camera
- Photo

Le moteur calcule alors les scores de pertinence (notamment via TF/TF-IDF et une mesure de similarité), puis affiche une liste de documents classés selon leur pertinence par rapport à la requête.

Enfin, pour les **TD8 à TD10**, le programme peut lancer l'interface **Streamlit** afin d'effectuer les recherches et l'exploration du corpus de manière plus interactive via un navigateur web.

Remarque : pour récupérer davantage de documents ou changer de sujet, il suffit de modifier les paramètres au début du script (ex. theme, limit, max_results) avant de relancer le programme.

IV. Validation et tests

La validation du projet a été réalisée à travers une combinaison de **tests unitaires** et de **tests globaux**, effectués progressivement au fil des travaux dirigés.

IV.1 Tests unitaires

Les tests unitaires ont été réalisés de manière progressive tout au long du développement du projet, principalement de façon **manuelle**, lors de l'exécution des différents scripts Python correspondant aux travaux dirigés (TD3 à TD10).

Chaque fonctionnalité a été testée individuellement afin de vérifier son bon fonctionnement avant son intégration dans l'ensemble du projet. Les principaux tests unitaires réalisés sont les suivants :

- Tests des fonctions de récupération des données (TD3) :
Les fonctions interrogant les APIs Reddit et arXiv ont été testées en vérifiant :
 - le nombre de documents récupérés,
 - la présence du texte,
 - la cohérence des métadonnées (titre, auteur, date, URL).
- Tests de sauvegarde et de chargement du corpus :
Les fonctions de sauvegarde dans des fichiers CSV et de rechargement du corpus ont été

testées afin de s'assurer que les données sauvegardées étaient correctement relues et qu'aucune information n'était perdue entre deux exécutions.

- **Tests des classes orientées objet** (TD4-TD5) :
Les classes *Document*, *RedditDocument* et *ArxivDocument* ont été testées lors de la création d'objets, de l'accès aux attributs spécifiques (nombre de commentaires, co-auteurs) et de l'héritage depuis la classe *Document*. La classe *Author* a également été testée en vérifiant l'ajout de documents et le calcul de la taille moyenne des textes.
- **Tests de la classe Corpus** :
Les méthodes d'ajout de documents, d'affichage, de statistiques par auteur, ainsi que la sauvegarde et le rechargement du corpus ont été testées individuellement pour vérifier la cohérence des données manipulées.
- **Tests des fonctions d'analyse textuelle** (TD6) :
Les fonctions de recherche par motifs (expressions régulières), de concordancier et de calcul des statistiques de fréquence ont été testées sur différents mots-clés afin de vérifier la pertinence des résultats retournés.
- **Tests du moteur de recherche** (TD7) :
La classe *SearchEngine* a été testée en lançant différentes requêtes utilisateur et en vérifiant que les documents retournés étaient correctement classés selon leur score de pertinence.

Ces tests unitaires ont permis de valider chaque composant du programme de manière isolée avant leur intégration complète, limitant ainsi les erreurs lors des phases finales du projet.

IV.2 Tests globaux

Les tests globaux ont consisté à exécuter le programme dans son ensemble, depuis le lancement du script principal jusqu'à l'affichage des résultats finaux.

Plusieurs cas ont été testés :

- Lancement du programme avec un **corpus déjà existant** (chargement depuis un fichier CSV)
- Lancement sans corpus existant, entraînant une **récupération automatique des données** via les APIs
- Saisie de requêtes contenant un ou plusieurs mots-clés
- Saisie de requêtes ne correspondant à aucun document pertinent

- Test de l'interface **Streamlit** (TD8 à TD10), incluant le chargement de fichiers CSV, la recherche interactive et l'exploration du corpus.

Ces tests ont permis de valider le bon enchaînement des différentes étapes du programme et la cohérence des résultats affichés à l'utilisateur.

V. Maintenance et perspectives

Le projet a été conçu de manière **modulaire**, ce qui facilite sa maintenance et son évolution. Chaque fonctionnalité est regroupée dans un fichier ou une classe spécifique, ce qui permet de modifier ou d'ajouter des éléments sans impacter l'ensemble du programme.

Plusieurs évolutions sont envisageables :

- Ajouter de **nouvelles sources de données** (autres réseaux sociaux, sites spécialisés, bases de données textuelles)
- Améliorer les méthodes de **traitement du texte** (lemmatisation, suppression de mots vides plus avancée)
- Intégrer d'autres **mesures de similarité** ou des techniques plus avancées de recherche d'information ;
- Enrichir **l'interface Streamlit** avec davantage de visualisations et d'options de filtrage
- Permettre à l'utilisateur de choisir dynamiquement les paramètres de récupération (thème, nombre de documents) directement depuis l'interface.

Ces évolutions seraient relativement **faciles à mettre en œuvre** grâce à la séparation claire entre le cœur du moteur, la gestion des données et l'interface utilisateur. Certaines améliorations plus avancées pourraient **toutefois nécessiter un temps de développement supplémentaire et l'utilisation de bibliothèques spécialisées**.

Conclusion

Ce projet a permis de mettre en œuvre l'ensemble des notions abordées durant le module de programmation Python, depuis la manipulation de données textuelles jusqu'à la conception d'un moteur de recherche simple et fonctionnel. Il a également permis de travailler sur la structuration du code, la programmation orientée objet et la mise en place d'algorithmes de recherche d'information.

Il a aussi constitué **une première expérience dans le développement d'une interface utilisateur en Python**, notamment à l'aide de la bibliothèque Streamlit. La mise en place de cette interface a permis de rendre le moteur de recherche plus accessible et plus interactif, tout en facilitant l'exploration des corpus. Cette approche a été particulièrement appréciée, car elle permet de visualiser concrètement le fonctionnement du programme et d'améliorer l'expérience utilisateur.

Enfin, ce projet constitue une base solide pour des développements futurs et une première introduction concrète à la conception d'applications complètes en Python.