IUT Villetaneuse
Université Sorbonne Paris Nord

ÉTUDIANT : THARINDU PERERA

Rapport de stage

Urssaf
Caisse nationale

PYTHON DANS LE BIG DATA

Tuteur de stage : WALIKANNAGAE Sheron
Enseignant référent : TORTI Agathe

MARS-JUIN 2025

Remerciements

Je tiens à adresser mes remerciements à celles et ceux qui ont contribué à l'obtention de mon stage jusqu'à l'aboutissement de celui-ci et qui m'ont également aidé lors de la rédaction de ce rapport.

Tout d'abord, je tiens à remercier mon responsable et tuteur de stage M. Sheron WALIKANNAGAE ainsi que sa supérieure Mme. Louise PHAM de m'avoir fait confiance et accorder l'opportunité d'effectuer un stage au sein de l'entreprise mais aussi pour leur accueil, disponibilité, pédagogie et enfin la qualité de l'encadrement au sein de l'équipe.

Je remercie également Mme. Agathe TORTI, Responsable des stages, pour les différents conseils apportés ainsi que son suivi

Sans oublier mes professeurs de l'IUT Sorbonne Paris Nord que je souhaite également remercier, pour l'éducation, les connaissances et le savoir-faire transmis tout au long de ces deux années de BUT qui m'ont été plus que bénéfiques et nécessaires au bon déroulement de mon stage.

Enfin je tiens à remercier Mme Sylvie CARDOSO pour son aide et précieux conseils concernant les entretiens et la rédaction de curriculum vitae / lettres de motivations durant les diverses séances en amphithéâtre.

SOMMAIRE

INTRODUCTION

En 2025, afin de valider mon BUT Informatique à l'Institut Universitaire de Technologie de Villetaneuse, j'ai dû chercher un nouveau stage ayant pour but de me mettre en situation professionnelle afin d'enrichir mes connaissances acquises durant la totalité du cursus. En effet, il s'agit d'une formation pluridisciplinaire et professionnalisante qui lie des compétences à la fois techniques comme la programmation et des compétences transversales telles que la gestion de projet et la communication.

Au cours de ce stage, j'ai pu avoir une nouvelle approche du monde professionnel de l'informatique différente de celle que j'ai fait durant mon stage de deuxième année. En qualité de stagiaire, j'ai occupé le poste de Développeur. Mon but était créer un programme ayant pour but de corriger et fiabiliser des adresses postales afin de pouvoir les rendre utilisables dans d'autres programmes

Lorsque j'ai intégré l'entreprise, l'entreprise avait déjà eu l'idée de faire ce programme mais celui-ci n'avait pas abouti. On a donc pu m'aiguiller dans les étapes à suivre afin de mener ce projet à bien, je ne partais donc pas totalement sans informations. Pour faire ce programme j'avais le droit d'utiliser Python, maîtrisant bien ce langage grâce à mes divers projets en première année de BUT ainsi que mon stage de deuxième année. En revanche, j'ai du faire un certain travail de recherche au préalable afin de sélectionner les bibliothèques python nécessaire, il a donc fallu entreprendre une auto formation en consultant des documentations en ligne, visionner des vidéos Youtube afin de bien choisir ce dont j'ai besoin pour répondre au mieux aux attentes de mon maître de stage.

Donc durant ces semaines, j'ai pu redécouvrir et mieux comprendre les notions de développement en Python, et apprendre de nouvelles notions qui sont les autres bibliothèques utilisées pour créer ce programme.

1. Présentation de l'entreprise

1.1 URSSAF

L'URSSAF (Union de Recouvrement des cotisations de Sécurité Sociale et d'Allocations Familiales) fait partie du réseau de la Sécurité sociale française. Sa mission principale est de collecter les cotisations sociales auprès des entreprises(employeurs) , en effet tous les salariés (hors secteur agricole) cotisent aussi auprès de l'Urssaf par l'intermédiaire de leur employeur. Que vous soyez en alternance, intérimaire, en CDI ou CDD, vous contribuez au financement de la protection sociale. Les cotisations et contributions sociales financent de nombreuses prestations et infrastructures sociales dans différents domaines (voir image 1 pour plus d'information) tels que :

- **La santé** : les rendez-vous médicaux, les arrêts de travail, les congés maternité et paternité, etc.
- **La retraite** : la retraite de base et l'allocation de solidarité aux personnes âgées.
- **La famille** : les allocations familiales, les aides au logement (APL etc.)
- **Le chômage** : les allocations chômage, l'aide à l'obtention du permis de conduire, l'aide à la mobilité.

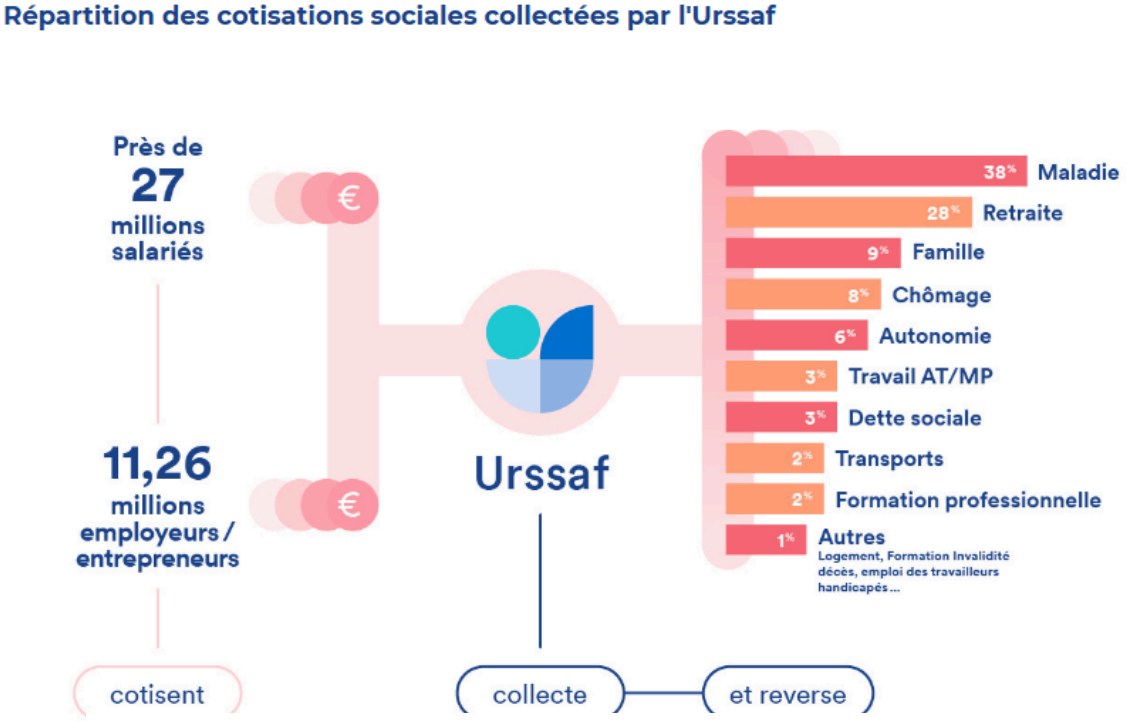


Figure 1 : Répartition des cotisations sociales de l'URSSAF

L'URSSAF est dirigé au niveau national par l'ACOSS (Agence centrale des organismes de sécurité sociale). L'ACOSS joue un rôle de coordination, d'orientation stratégique et de gestion de la trésorerie pour l'ensemble du réseau URSSAF. En parallèle, les URSSAF locales assurent un rôle opérationnel auprès de chaque région ou département.

Mon stage s'est déroulé à l'URSSAF à Montreuil, qui concentre des équipes de différents pôles de l'entreprise. Le site regroupe plusieurs directions, dont celles en charge de projets numériques, de la data.

1.2 Organigramme

L'ACOSS est structurée autour de plusieurs directions, dont les directions du numérique et de la data, qui regroupent des pôles projets. Mon stage s'est déroulé au sein d'un de ces pôles : celui du Big Data (Le Big Data est l'ensemble des données massives qui nécessitent des outils spécifiques pour être stockées, traitées et analysées).

L'image ci-dessous représente une partie de l'organigramme de l'URSSAF avec mon point de vue, montrant les principales personnes avec qui j'ai pu interagir pendant mon stage. On y retrouve plusieurs membres des directions DSI (Direction des Systèmes d'Information) et SDED (Sous-Direction Études et Décisionnel), principalement rattachés aux pôles décisionnel et big data. Cependant à préciser que je n'ai pas interagi avec les personnes au dessus de Mr NYEMBO Herbert car ils ne sont pas présent sur le site où je me trouvais et sûrement bien trop occupé de par leur position dans l'entreprise.

Certaines de ces personnes ont eu un rôle direct dans le suivi ou la relecture de mon travail, comme Mr WALIKANNAGAE Sheron, mon maître de stage, qui m'a encadré techniquement, ou un de ces collègues avec qui je travaillais également mais qui n'est pas dans l'organigramme car il est dans une direction différente.

Ce type d'environnement m'a permis de m'adapter au fonctionnement d'un vrai projet en entreprise, même si je travaillais majoritairement en autonomie.

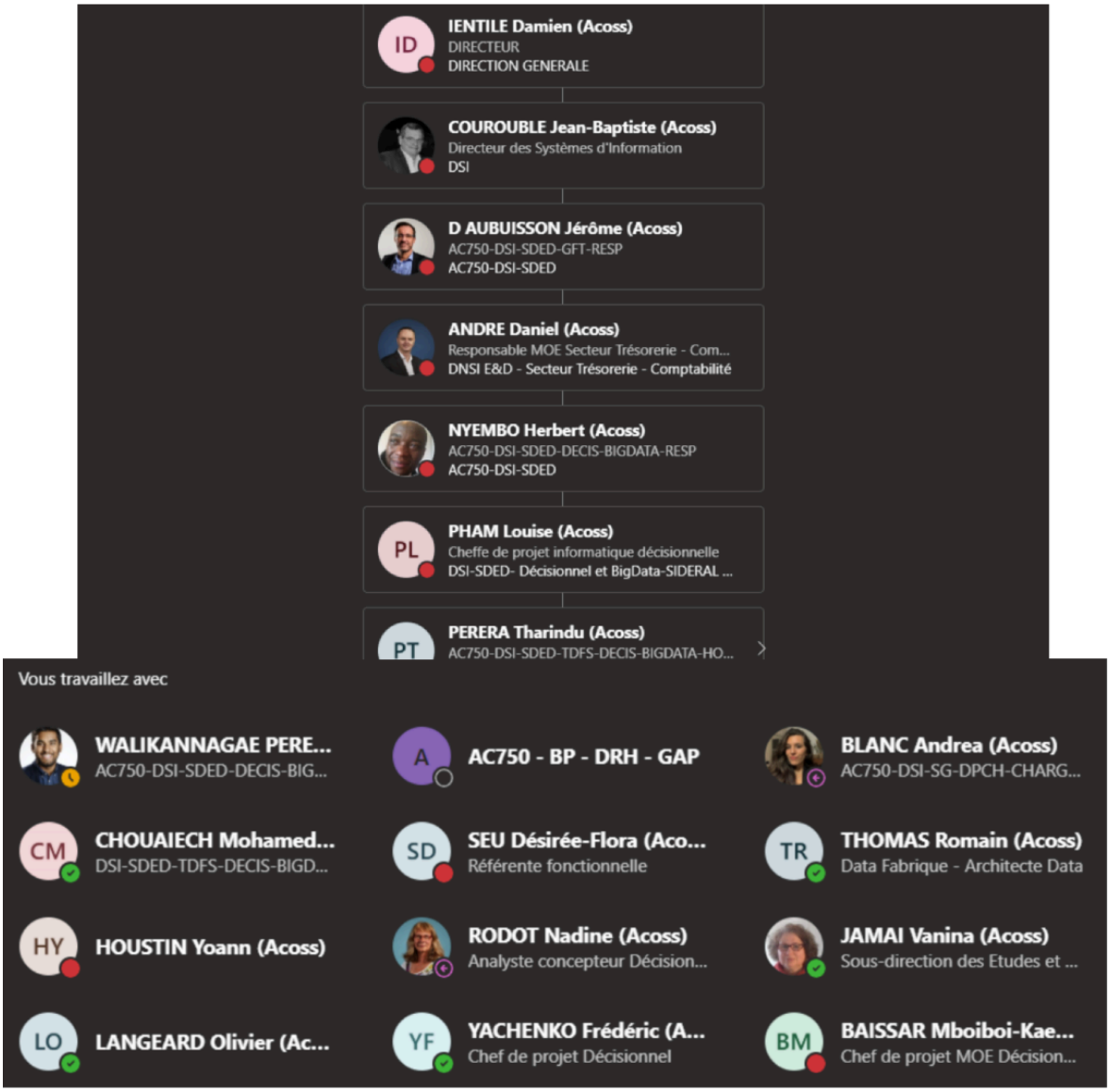


Figure 2 : Organigramme du pôle big data URSSAF

1.3 Le but du projet pour l'entreprise ?

L'objectif de la mission confiée s'inscrit dans une démarche dans un premier temps de la fiabilisation des données utilisées dans des traitements internes et dans les échanges entre les services de l'URSSAF. Les adresses postales saisies par les usagers ou les agents sont souvent erronées, mal structurées ou non conformes car pour le moment la méthode d'écriture de ces

adresses reste dans un format libre car il est compliqué de définir de nouvelles règles d'écriture des adresses et les rendre fonctionnel dans toute la France.

Ce projet vise à mettre en place une chaîne de traitement automatique capable de :

- Les normaliser dans un format exploitable,
- Et les faire correspondre automatiquement à des adresses officielles.
- Et récupérer coordonnées GPS des adresses (x,y longitude et latitude)

L'objectif à terme est de garantir la qualité des données utilisées dans les projets comme celui de HORUS

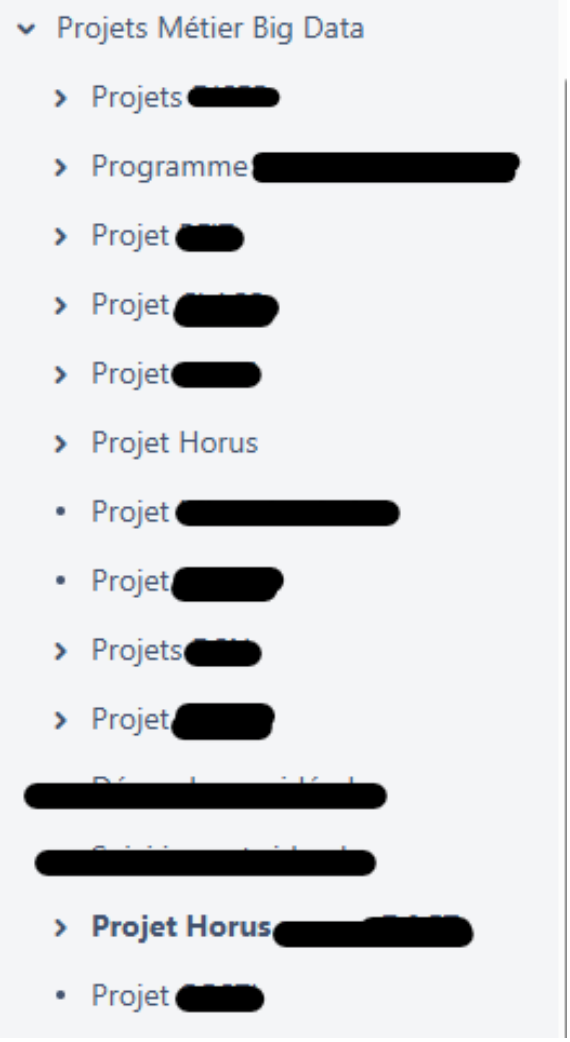


Figure 3 : Projet big data

Comme vous pouvez le voir dans la figure 3, il y a beaucoup de projets en cours dans l'équipe. et celui qui nous intéresse est le projet HORUS qui a pour objectif d'améliorer la détection des revenus issus des plateformes collaboratives (Airbnb, Uber ou encore Vinted). Avec la croissance de l'économie numérique, beaucoup d'utilisateurs perçoivent grâce à ces plateformes des revenus qui dépassent les seuils autorisés sans pour autant se déclarer comme travailleur indépendant, ce qui normalement est obligatoire passé un certain seuil de revenu.

L'enjeu de l'URSSAF est donc de mettre en place un système capable d'identifier automatiquement les personnes dépassant les seuils grâce aux données transmises par ces plateformes. Ensuite contacter et accompagner les utilisateurs à se régulariser.

Il existe également un autre projet appelé ARTIC, dans lequel ma mission prend tout son sens. Ce projet porte sur le calcul des réductions fiscales accordées à certains employés. Lorsqu'un salarié se trouve à une certaine distance de son lieu de travail, il peut sous certaines conditions, bénéficier d'une exonération fiscale partielle. Par exemple, si la distance dépasse 50 km, le salarié peut avoir une exonération avec un pourcentage X.
Cependant, pour éviter les abus ou déclarations erronées, l'URSSAF souhaite automatiser la vérification de ces distances en géolocalisant les adresses déclarées à l'aide des coordonnées GPS disponible sur internet. Grâce à ça, le système pourra déterminer si le salarié est réellement éligible à une réduction.

2. Gestion de projet

2.1 Méthodologie Agile

Maintenant parlons de la gestion de projet; durant mon stage, j'ai évolué dans un environnement structuré selon la méthode "Agile", utilisée dans l'ensemble des projets de la direction. Cette approche est basée sur des cycles courts avec des améliorations progressives à chaque fois qu'une tâche est terminée, permettant de livrer des fonctionnalités par étapes, en intégrant les retours et les ajustements que l'on pouvait me faire durant le développement de mes programmes.

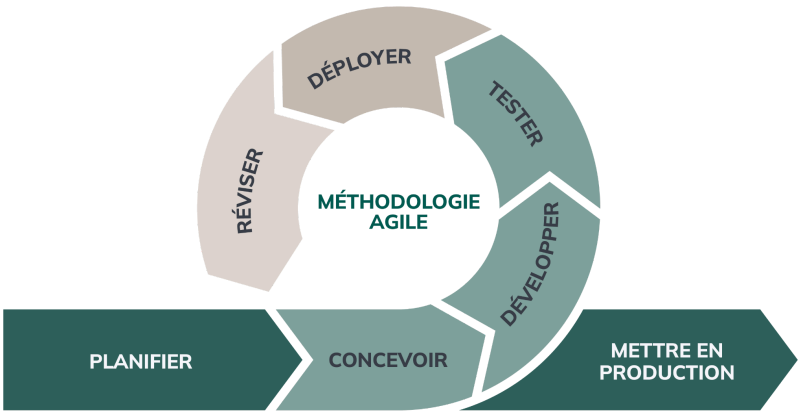


Figure 4 : Schéma méthode agile

La méthodologie Agile repose sur plusieurs cycles (voire figure 4 pour mieux comprendre), chacun intégrant les phases suivantes :

- Planification : La première partie et la plus importante, la définition des tâches à réaliser, en concertation avec les équipes. Cette partie permet de mettre tout le monde d'accord sur ce qu'il faut faire avant de débuter.
- Conception et développement : Cette partie consiste en la mise en œuvre des tâches techniques prévues, ici la programmation des algorithmes

- Tests : Ici on va vérifier si le code créer dans la partie précédente fonctionne correctement grâce à des jeux de tests ou autre pour vérifier si les résultats attendus sont bien là comme voulu

- Déploiement : Il s'agit de la mise à disposition progressive des résultats.

- Révision : Si on trouve divers bugs ou des moyens de faire des améliorations, avant de relancer un nouveau cycle.

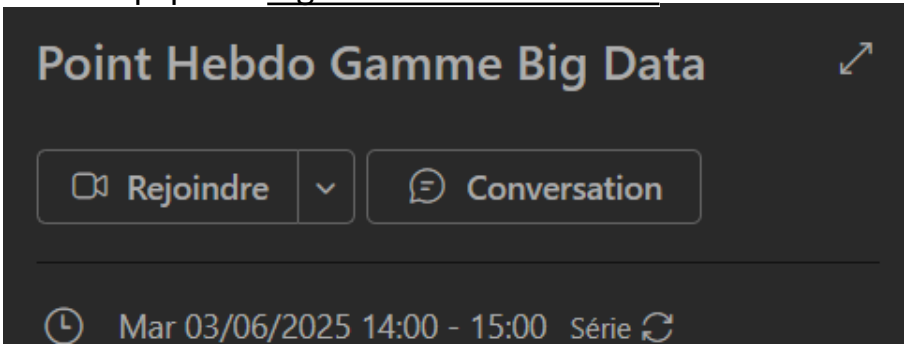
Et une fois tout terminé nous pouvons lancer la mise en production.

Cette méthode de travail m'a permis de mieux m'organiser, d'adopter une logique dans le développement de mes scripts, et de bénéficier de retours réguliers de la part de mon tuteur et ses collègues concernant les éventuelles ajouts ou modifications à faire. C'est pour cette raison que j'ai suivi cette méthode durant l'intégralité de mon stage. À chaque fonctionnalité ou correction, je faisais des tests dans un fichier isolé avant de l'ajouter au reste du projet afin de ne pas créer de soucis, ensuite je démarrais le programme entier pour voir si tout fonctionnait correctement avant de continuer. Si ce n'était pas le cas je repassais par la phase de tests avec mes programmes de tests créé exprès. En effet, les jeux de tests sont importants afin de garantir la qualité du code.

2.2 Réunions

En plus de cette méthode de travail spécifique, il faut savoir que chaque semaine nous faisions des réunions (voir figure 5) avec toutes les personnes de l'équipe Big Data (à savoir environ 25 personnes répartis dans la france entière sur divers sites) Ces diverses réunions étaient, à mon sens, très intéressante car elle permettait de mettre en commun toutes les choses qu'il y avait à dire, que ce soit les avancées, les points bloquants ou autre et ceux sur tous les projets en cours dans cette équipe.

Figure 5 : Réunions Teams



Par ailleurs, j'ai eu la chance d'assister à un séminaire qui a lieu chaque trimestre ,organisé avec toutes les personnes de la gamme big data. Ils sont conviés à venir sur place dans les locaux de l'URSSAF de Montreuil afin d'échanger le temps d'une journée entière permettant ainsi de renforcer les liens entre tous les membres de l'équipe ce qui est, je trouve, extrêmement important et ce n'est pas quelque chose à négliger

3. Missions et développement

3.1 Environnement de travail

Durant mon stage, j'ai été intégré dans une équipe constituée de profils techniques (développeurs, data analysts). L'environnement de travail au sein du site de Montreuil (qui est séparé en 2 bâtiments appelés "Wii" et Gaumont), possède des locaux spacieux, des bureaux partagés et des espaces communs favorisant les échanges. L'accueil a été bienveillant, et l'équipe s'est montrée disponible pour m'accompagner dans la compréhension des outils et des enjeux.

Lors de mon premier jour on m'a donc dit que l'équipe que j'allais rejoindre se trouve dans le bâtiment "Wii". Suite à quoi avec mon maître de stage nous nous sommes rendus dans l'autre bâtiment afin que je puisse récupérer l'ordinateur que l'on a mis à ma disposition.

Figure 6 : Batiment "Wii"

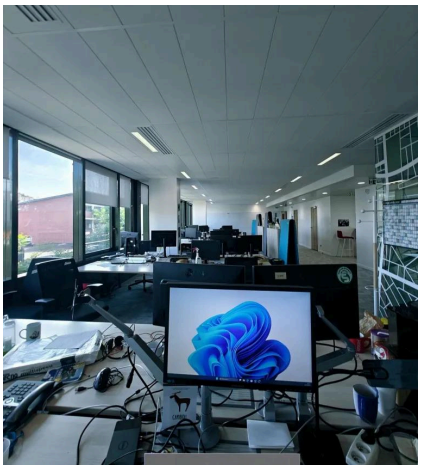


Figure 7 : Locaux

Celui-ci est très sécurisé puisqu'en plus d'un mot de passe solide, il possède aussi un Bitlocker (un outil de sécurité intégré à Windows qui sert à chiffrer le contenu d'un disque dur) pour renforcer la sécurité et éviter que des données sensibles puissent être volées.

De plus, pour pouvoir accéder aux diverses applications/réseaux internes de l'URSSAF il faut se trouver en présentiel dans le bâtiment et/ou avoir une carte à puce attitrée (développée par Thales) permettant

de se connecter à un VPN et donc accéder à tout ce qui est nécessaire pour travailler.

Ensuite en revenant, j'ai bénéficié d'un poste de travail dédié (voir figure 7), avec les accès aux ressources techniques nécessaires pour travailler dans les meilleures conditions possibles.

Concernant les horaires de travail, ils sont libres tant que l'on fait les horaires journaliers (7h par jour pour mon cas) une fois connecté au réseau interne de l'URSSAF on peut se connecter à un outils qui nous permet de badger nos entrées et sorties (pauses déjeuner incluses). Cependant il y a quand même des règles à respecter qui sont les suivantes : Interdit de Badger avant 7h et après 20h pour respecter le droit à la déconnexion ainsi que certains horaires durant lesquels nous sommes obligés d'être en train de travailler (sur site ou distanciel). De plus nous pouvons déposer nos jours de télétravail comme on le souhaite sauf le jeudi qui est le jour où le présentiel est obligatoire pour les gens travaillant sur le site de Montreuil.

3.2 Choix technologiques

Pour répondre aux besoins du projet, plusieurs choix technologiques ont été faits :

Langage : Python a été utilisé pour la majorité du traitement, en raison de sa richesse en bibliothèques pour le traitement de texte et la manipulation de données. De plus ce langage est très utilisé à l'URSSAF donc on m'a recommandé

Bibliothèques principales :

- Rapidfuzz : Elle intègre une version rapide de l'algorithme de Levenshtein

- Pandas : Pour la manipulation des fichiers CSV fournis par l'URSSAF.

- Re (=expressions régulières) : Pour le nettoyage des adresses.

- Matplotlib : pour la visualisation des résultats et des données statistiques afin de voir les améliorations du programme.

- Geopy : pour le géocodage d'adresses, en complément pour vérifier la cohérence avec la base d'adresse nationale disponible sur internet.
- Dask : pour effectuer du traitement parallèle, utile pour manipuler de gros volumes de données.

En parallèle, j'ai aussi envisagé d'utiliser la bibliothèque Libpostal (un outil de parsing d'adresses open source pour nettoyer facilement les adresses mais je me suis rendu compte que ce n'était pas assez fiable donc je ne l'ai pas gardé longtemps),

4. Réalisations

Dans cette partie je vais parler non seulement de tout ce que j'ai réalisé durant le stage mais aussi montrer le cheminement de pensées qu'il y a eu afin de répondre au mieux aux attentes de mon tuteur de stage et ses collègues. J'évoquerais aussi les difficultés rencontrées durant mes travaux.

À chaque fois avant de démarrer un nouveau programme je faisais un schéma afin de savoir ce que le programme allait faire et dans quel ordre. Pour que les lecteurs de ce rapport puissent mieux visualiser le fonctionnement des programmes en plus des explications je vais rajouter ces schémas afin de rendre le tout plus compréhensible.

4.1 Génération d'erreurs simulées

La première mission qui m'a été donnée de réaliser pour ce stage était de développer une fonction permettant de générer des adresses erronées à partir d'adresses correctes.

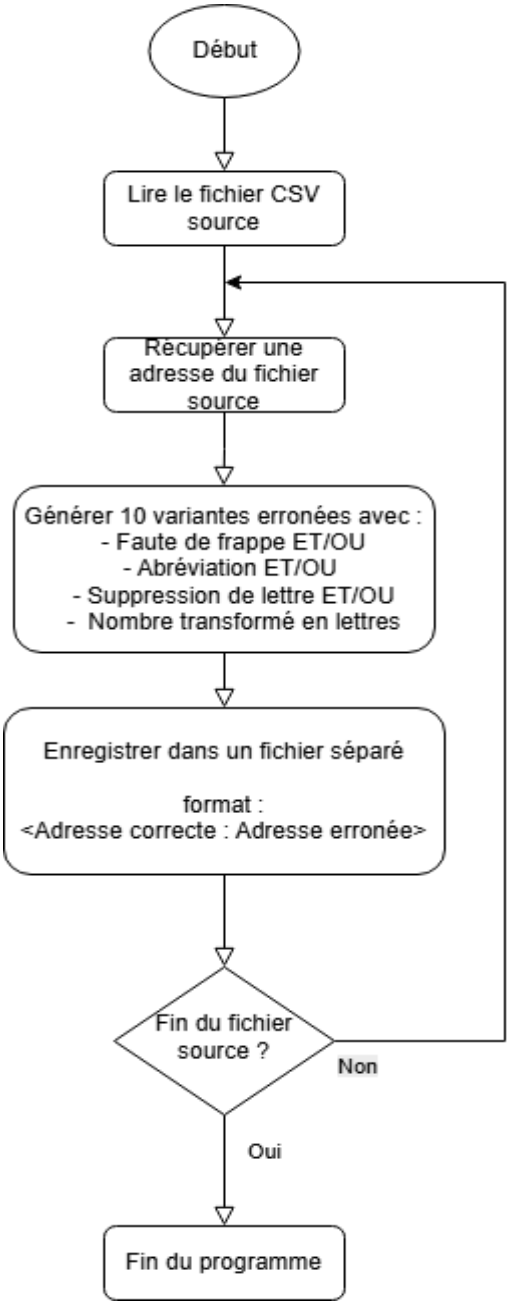
Ce générateur simule donc des erreurs que l'on retrouve fréquemment dans des saisies par des humains comme vous et moi, les erreurs pouvant apparaître sont les suivantes :

- **Fautes de frappe** : remplacement d'un caractère par une lettre voisine sur le clavier. (ex : rue → rye ou rie. Car la lettre u est à côté du y et du i)
- **Suppressions de caractères** : lettres manquantes dans un mot (ex : rue → re).

- **Ajouts de caractères** : insertion d'une lettre superflue avec une lettre proche de celle saisie (ex : rue → ruer. Le e et le r sont côte à côte sur les claviers).

Cela m'a permis de mieux comprendre les types d'erreurs les plus fréquents et leur impact même si l'on peut penser que cela n'est que minime dans la vie de tous les jours.

Voilà le premier schéma que j'ai réalisé avant de faire mon programme et ça sera pareil pour les suivants. (Il s'agit évidemment d'une version mise au propre)



Dans un premier temps, une fois le programme démarrer, on lis le fichier CSV fourni en paramètres celui-ci contient simplement des adresses (réelles ou non).

Ensuite le programme va prendre une adresse, puis choisir aléatoirement" (voir Annexe 1 de pour plus de détail sur le code) le nombre d'erreurs qu'il va faire et ensuite choisir parmi les 3 types d'erreurs évoqués plus tôt avant de les appliquer à l'adresse.

Une fois fait il va ouvrir un autre fichier CSV et y insérer l'adresse correcte avec ses variantes avec les fautes.

Si le 1er fichier CSV ouvert au départ est terminé alors le programme s'arrête sinon il passe à l'adresse suivante jusqu'à la fin du programme

Figure 8 : Schéma Algorithme d'erreurs

4.2 Constitution de la base de référence (BAN)

```
Adresse correcte:Adresse erronée
1  Rue de la Cie Paris Orleans, 94480 Ablon-sur-Seine:1 rur de la cje paris ormeans, 94480 ablon-sur-seine
1  Rue de la Cie Paris Orleans, 94480 Ablon-sur-Seine:un rue de la cie paros orleans, 94480 ablon-sur-seine
```

Figure 9 : Résultats de l'algorithme d'erreurs

Une fois le programme de simulation d'erreurs de frappes fonctionnel, on m'a confié un fichier de test pour pouvoir continuer à avancer. Celui-ci contient des adresses saisies par des salariés, que je devais nettoyer et fiabiliser automatiquement, je vais l'appeler" fichier URSSAF" pour la suite.

Pour comparer ces adresses avec des références fiables, j'ai téléchargé sur internet l'ensemble des adresses officielles françaises depuis le site de la Base Adresse Nationale (BAN). Cette base contient plusieurs millions de lignes, réparties par département, et peut être difficile à manipuler dans sa forme brute.

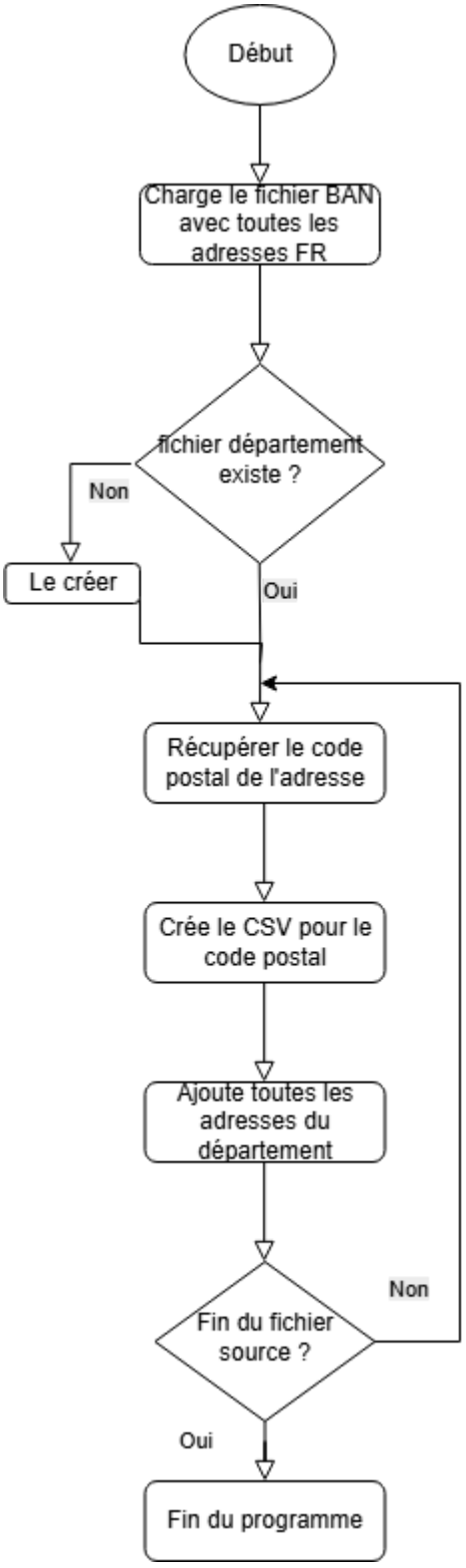
Étant donnée que je produis un algorithme de recherche qui va se lancer sur un fichier BAN possédant des millions de lignes ce n'est absolument pas optimisé de devoir l'ouvrir à chaque ligne d'adresse du fichier URSSAF.

Afin de régler ce problème et optimiser les performances du programme, j'ai conçu un script Python à part permettant de générer un fichier CSV par département existant dans le fichier BAN, contenant uniquement les champs écrient dans ce fichier à savoir dans l'odre :
id; id_fantoir; numero; rep; nom_voie; code_postal; code_insee; nom_commune; code_insee_ancienne_commune; nom_ancienne_commune; x; y; lon; lat; type_position; alias; nom_ld; libelle_acheminement; nom_afnor; source_position; source_nom_voie; certification_commune; cad_parcelles et enfin departement

```
id;id_fantoir;numero;rep;nom_voie;code_postal;code_insee;nom_commune;code_insee_ancienne_commune;nom_ancienne_commune;x;y;lon;lat;type_position;alias;nom_ld;libelle_acheminement
75103_ka7f7y_00001;;1;;Voie B/3;75003;75103;Paris 3e Arrondissement;;;652996.66;6862232.89;2.359369;48.858416;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
75103_ka7f7y_00002;;2;;Voie B/3;75003;75103;Paris 3e Arrondissement;;;653007.76;6862235.8;2.35952;48.858443;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
75103_ka7f7y_00003;;3;;Voie B/3;75003;75103;Paris 3e Arrondissement;;;653005.33;6862243.39;2.359486;48.858511;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
75103_ka7f7y_00004;;4;;Voie B/3;75003;75103;Paris 3e Arrondissement;;;653015.69;6862244.86;2.359627;48.858525;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
75103_ka7f7y_00005;;5;;Voie B/3;75003;75103;Paris 3e Arrondissement;;;653016.45;6862257.08;2.359636;48.858635;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
75103_ka7f7y_00006;;6;;Voie B/3;75003;75103;Paris 3e Arrondissement;;;653026.13;6862256.89;2.359768;48.858634;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
75103_ka7f7y_00007;;7;;Voie B/3;75003;75103;Paris 3e Arrondissement;;;653027.44;6862264.56;2.359785;48.858703;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
75103_ka7f7y_00001_s;;1;s;Voie B/3;75003;75103;Paris 3e Arrondissement;;;653000.93;6862234.3;2.359427;48.858429;parcelle;;;PARIS;VOIE B 3;commune;commune;1;;75
```

Figure 10 : exemple fichier BAN

Cette étape a permis de rendre les recherches plus rapides, en limitant les comparaisons, il ne prend et ouvre que les fichiers BAN nécessaires pour chaque adresse du fichier URSSAF en regardant le numéro du département.



Une fois le programme lancé, celui- ci va prendre le fichier ayant toutes les adresses de france à l'intérieur.

Ensuite vérifier si le fichier “dep” (pour département) existe afin d’enregistrer les futurs CSV dedans, si c’est pas le cas le code va en créer un.

Ce qu’il faut savoir avec la BAN c’est que les adresses sont déjà triés par département. Donc en sachant ça il est facile de récupérer uniquement la valeur correspondant au département de chaque ligne et ensuite de créer un CSV correspondant au département en question avant d’y mettre toutes les adresses

Une fois fait, on vérifie si nous nous trouvons bien à la fin du fichier et si ce n’est pas le cas, on cherche à la ligne suivante afin de créer un nouveau fichier XX.csv (XX étant les 2 premiers numéro du département) puis on répète l’opération en boucle jusqu’à la fin du fichier BAN

Figure 11 : Schéma Algorithme de découpage

4.3 Création de l'algorithme de correspondance

L'étape suivante a été le développement du cœur du projet : un programme capable de retrouver automatiquement l'adresse correcte dans la BAN, à partir d'une adresse URSSAF incomplète ou erronée ou même correcte évidemment.

Avant d'expliquer mon programme je dois parler d'une notion que j'ai appris durant mon stage servant à mesurer la similarité entre deux chaînes de caractères, il s'agit de l'algorithme de "Levenshtein", qui mesure le nombre minimum d'opérations nécessaires (insertion, suppression, substitution) pour passer d'une chaîne X à une chaîne Y (ici les chaînes sont les adresses). À la fin, l'algorithme donne un chiffre entre 0 et 1 ; plus le chiffre est proche de 1 plus l'adresse X ressemble à l'adresse Y et si c'est égale à 1 alors les deux adresses sont les mêmes.

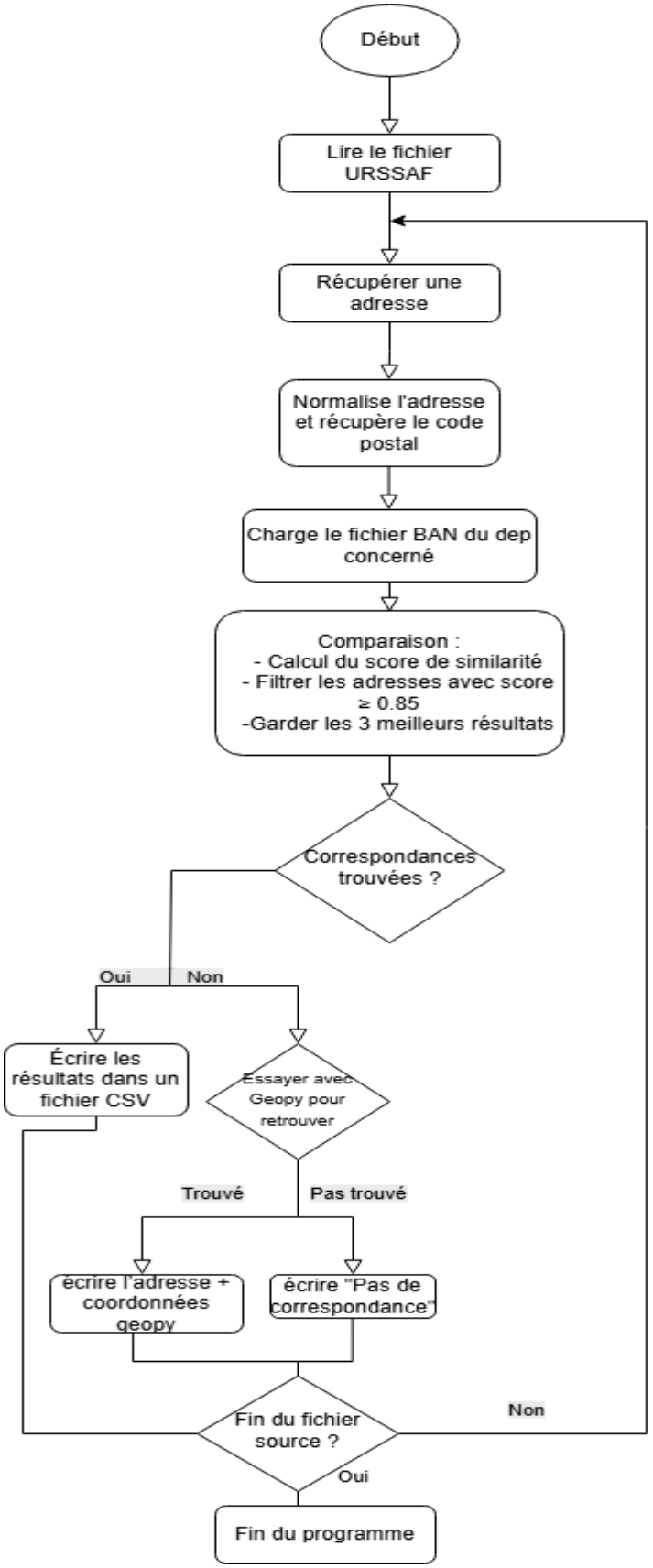
Pour un exemple concret, prenons le mot "chat" et "chats". La distance de Levenshtein sera de 1 car on a fait une insertion. Ensuite on prend la longueur maximal de la chaîne de caractères entre les deux. chat à une longueur de 4 et chats 5 donc on garde 5. Et enfin on fait 1 - distance/longueur maximal ce qui donne dans notre cas : 1 - ⅕ = 0.8. Donc le score de similarité sera de 0.8

Ensuite, comme dit plus tôt énormément d'adresses ne sont pas utilisables dans l'instant T dû au fait que les personnes écrivent leurs adresses n'importe comment rendant le nettoyage automatique assez difficile et c'est la raison principal de pourquoi vouloir faire un code de correction automatique des adresses afin que personne n'est besoin de repasser dessus derrière.

Pour ça il fallait prendre en compte beaucoup de facteurs comme par exemple le fait que certaines personnes peuvent écrire "bâtiment" ou "bat" voir "bât". Tant de manière d'écrire différente à prévoir pour de rendre l'adresse la plus "propre" possible afin de faire les comparaisons après. Pour ce faire j'ai dû créer des dictionnaires de mots où à chaque test je rajoutais de nouveaux mots parasites qui auraient pu interférer.
Il y avait beaucoup de Regex (expression régulière) différentes à créer car en fonction des mots supprimer le comportement ne sera pas forcément le même. En reprenant l'exemple de bâtiment, il faudrait supprimer uniquement la lettre

ou le numéro qui suit ce mot. Mais cela ne fonctionne pas de la même manière si on écrit “Chez” puisque la personne peut spécifier le nom ET le prénom de la personne donc il faut le prendre en compte pour le supprimer.

Voici la logique suivie pour la création de cet algorithme de correspondance, et j’y ai combiné plusieurs techniques :



Dans un premier temps je suis passé par la lecture du fichier URSSAF que l’on m’a fournis pour mes tests

Ensuite on nettoie une adresse du fichier URSSAF (suppression des éléments non pertinents, harmonisation des formats)

Puis on identifie le département à partir du code postal de l’adresse, avant de charger le fichier BAN correspondant

Pour ensuite comparer l’adresse nettoyée à toutes les adresses de ce département en utilisant “rapidfuzz”

Une fois fait on retient jusqu’à trois adresses BAN parmi les plus proches, à condition qu’elles atteignent un score supérieur ou égal à 0.85

Si on ne trouve pas de correspondance on test avec l’API Geopy pour geolocaliser les adresses et si c’est toujours pas le cas on écrira “pas de correspondance”

Et pour terminer on écrit les correspondances retenues ou non dans un fichier CSV dédié qui servira plus tard pour d’autres projets de l’URSSAF.

Figure 12 : Schéma Algorithme de correspondance

Ce fichier de résultats contient les éléments suivants :

- Un numéro de séquence, utile si plusieurs correspondances sont trouvées pour une même adresse (permet un traitement plus simple dans une base de données),

- L'adresse d'origine (issue de l'URSSAF),

- L'adresse correspondante issue de la BAN,

- Les coordonnées géographiques de l'adresse BAN (x, y, longitude, latitude) OU les coordonnées trouvées grâce à geopy

- Le score de correspondance calculé par rapidfuzz.

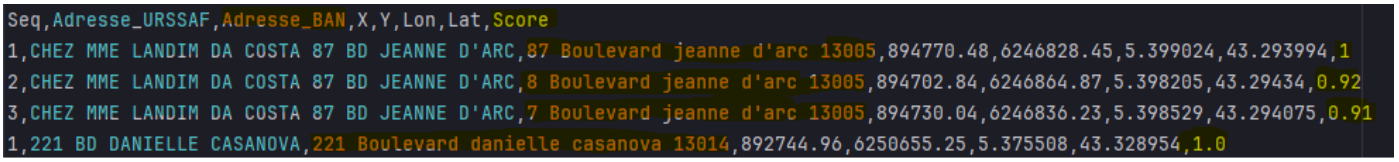


Figure 13 : Score de similarité

Dans la figure ci-dessus on peut voir ce que le fichier de résultats après un passage dans le programme de calcul de similarité. On y voit bien en bleu une adresse rentrée par un employé de l'URSSAF puis en orange après un nettoyage de l'adresse l'algorithme à réussi à trouver l'adresse dans toutes celles du département avant de donner un score de 1 car les deux adresses sont les mêmes. Puis on peut voir d'autres similarités trouvées mais le score correspondant est plus bas car on peut voir qu'il y a des différences (à savoir le numéro qui n'est pas le même). Les trois adresses ayant le plus haut score ont été mises dans le fichier on passe donc à une nouvelle adresse et comme on peut le voir le numéro de séquence est bien retourné à un.

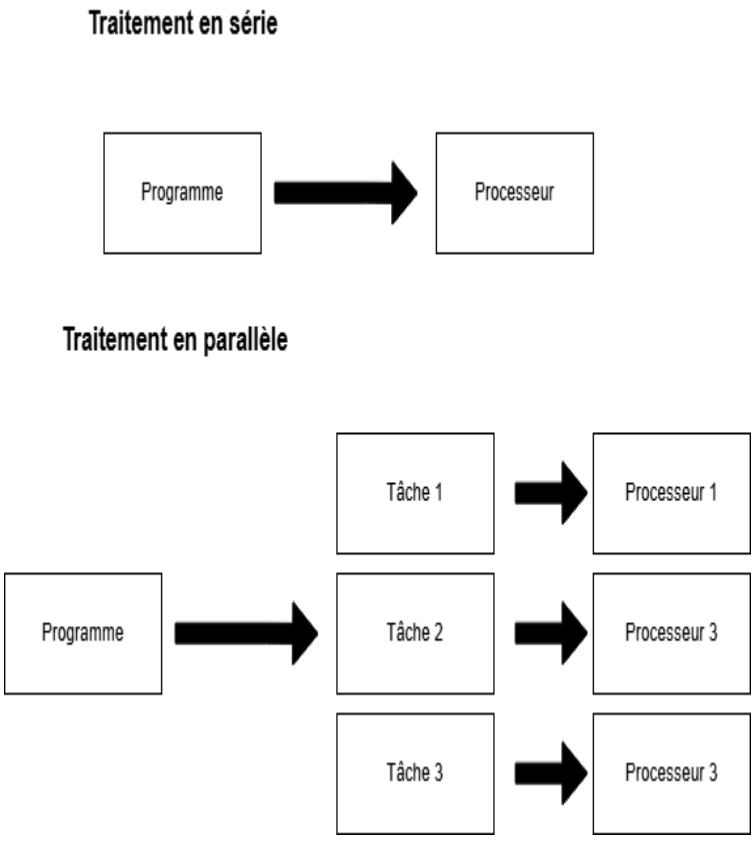
Évidemment ce que je viens de présenter est la version plus ou moins finale du programme mais pour arriver à ce résultat je suis passé par beaucoup d'étapes et tests différents.

Par exemple, au départ je l'avais reproduit l'algorithme de Levenshtein sur python moi même à partir du modèle disponible sur Wikipedia. Après l'avoir réalisé on s'est rendu compte que celui-ci était beaucoup trop lent, en effet pour rechercher environ 5 adresses présent dans un même département bien que cela fonctionnait, il mettait plus de 15 minutes à faire la tâche. Après avoir cherché plein de moyen différent pour voir comment l'améliorer nous avons décidé de remplacer mon algorithme par celle de la bibliothèque du nom de "rapidfuzz", beaucoup plus performante. Cette dernière est écrite en C++ en langage plus performant que python et il s'agit d'une version optimisée pour le calcul de similarité. Elle permet un traitement plus rapide, notamment grâce à une meilleure gestion de la mémoire et à des algorithmes adaptés aux comparaisons massives, ce qui réduit considérablement le temps de traitement. En comparaison, pour retrouver ces 5 mêmes adresses, cela ne prenait que quelques secondes avec rapidfuzz.

De plus, comme je l'ai dit précédemment Levenshtein nous renvoie une valeur que je vais appeler score de similarité. Pour garder une certaine cohérence et garantir les résultats le plus ressemblant possible nous avons décidé, avec un de mes collègues, de mettre un seuil minimal de 0.85 pour filtrer les correspondances. Car en dessous les adresses ressemblent de moins en moins à celles recherchés et on s'éloigne donc trop d'un résultat fiable rendant la futur exploitation des données incertaines. En plus de cela, nous ne gardons que 3 adresses au cas où il y en aurait trop au-dessus du seuil pour ne pas surcharger la mémoire des ordinateurs.

Ensuite on s'est rendu compte qu'étant donné que les fichiers URSSAF fournis sont assez gros (plus de Onze milles adresses à tester) cela pouvait affecter la puissance de l'ordinateur de donc pour pallier à ce problème j'ai fait un travail de recherche pour trouver des moyens d'optimiser au mieux quand je suis tomber sur une bibliothèque python appelé Dask qui est bibliothèque spécialement conçue pour le traitement parallèle de données. Elle permet de découper un traitement lourd en plusieurs petites tâches exécutées simultanément, ce qui améliore considérablement la vitesse d'exécution, tout en limitant l'impact sur la mémoire.

Voilà un schéma réalisé par mes soins (voir figure 14 ci dessous) montrant la différence entre un traitement d'un programme classique et un en utilisant dask et le traitement en parallèle



Grâce à son intégration, j'ai pu diviser le fichier d'adresses BAN en plusieurs partitions et traiter chaque adresse URSSAF en parallèle, ce qui rend le programme beaucoup plus fluide même avec un grand nombre d'adresses puisque les tâches sont divisés pour être traiter en même temps

Figure 14 : Différence traitement parallèle traitement en série

Les autres difficultés rencontrés concernaient essentiellement en la création des Regex qui devaient être correct afin d'avoir le bon comportement sur l'adresse et ne pas supprimer sans le vouloir des données importantes au lieu de ce qui n'est pas utile. Je vais donc expliquer les étapes du nettoyage de l'adresse. Vous pourrez vous référer à l'annexe pour mieux voir le code que je vais décrire tout de suite.

Lignes 7 à 19 : Un dictionnaire d'abréviations est défini pour y mettre des types de voies (bd ou av..) avec leurs versions complets (boulevard, avenue...). sachant qu'il n'y a pas de raccourcis dans la ban il est important de le corriger.

Ligne 21 : Début de la fonction normaliser_adresse, qui prend en entrée une adresse.

Ligne 22 : L'adresse est convertie en minuscules et les espaces en début ou fin de chaîne sont supprimés avec strip().

Ligne 24 : Tous les enchaînements d'espaces (tabulations, doubles espaces, etc.) sont remplacés par un simple espace.

Ligne 26 : On extrait le code postal à l'aide d'une expression régulière qui recherche une suite de 4 ou 5 chiffres à la fin de l'adresse (car il est arrivé que les gens ne mettent que 4 chiffres si leur code postal est entre 1 et 9).

Ligne 27 : Si un code postal est trouvé, il est stocké dans la variable code_postal. Sinon, la valeur est une chaîne vide.

Ligne 29 : Une seconde expression régulière est utilisée pour supprimer temporairement le code postal de l'adresse. Cela évite d'interférer avec les prochaines étapes de nettoyage.

Ligne 31 : Les accents sont supprimés à l'aide de la normalisation Unicode (NFD), ce qui transforme les caractères accentués (comme é) en une version non accentuée (e). On encode ensuite en ASCII pour filtrer uniquement les caractères standards et ne pas avoir de caractère étrange comme .

Ligne 33 : On supprime les mentions telles que "chez", "mme", "mr" etc, ainsi que tout ce qui suit jusqu'au premier chiffre (souvent un numéro de rue). L'expression régulière utilise (?=\d) pour ne s'arrêter qu'avant une suite de chiffres.

Ligne 35 : On applique une règle similaire, mais cette fois-ci au cas où la mention "chez" ou "mme" apparaîtrait en milieu d'adresse.

Ligne 37 : Une regex est utilisée pour supprimer tout ce qui est "etg", "apt" (et tout ce qui il y a écrit entre les parenthèses) qui suivies d'un chiffre ou d'une lettre (ex. : "étage 3", "apt A"). On peut y voir plusieurs abréviations pour la même chose mais c'est parce que les utilisateurs écrivent tous de manière différente.

Lignes 39 à 40 : Le dictionnaire défini plus tôt est utilisé pour remplacer chaque abréviation par sa version longue.

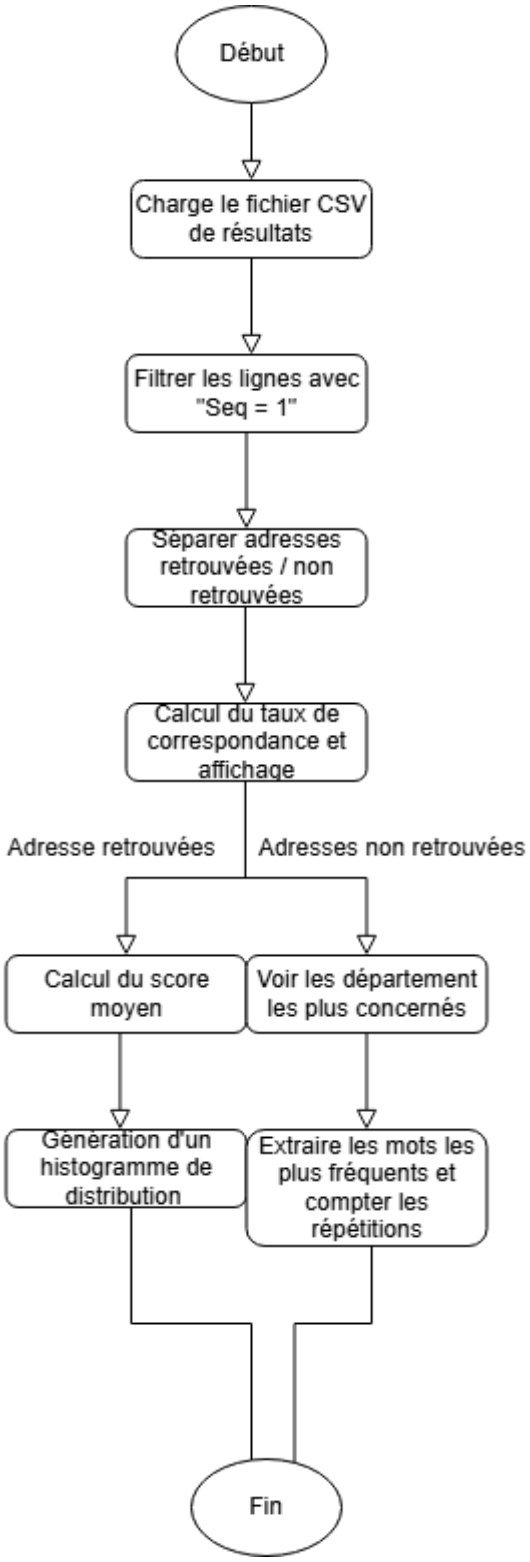
Ligne 42 : Un dernier nettoyage est effectué pour ne garder qu'un seul espace entre les mots. On s'assure aussi que l'adresse ne contient pas de doublons d'espaces ou d'espaces parasites qui seraient apparus avec toutes les corrections faites.

Ligne 43 : Enfin, on reconstruit l'adresse finale, en y ajoutant le code postal retiré plus tot.

Tous ces types de corrections ont été ajoutés petits à petit en faisant des tests avec le fichier fournis par l'URSSAF et en trouvant à chaque fois de nouveaux types d'écriture.

4.4 Algorithme de statistiques

Cet algorithme avait été créé durant la phase de test et d'amélioration du code présent de la partie juste avant. J'avais eu l'idée de le créer surtout pour but de voir l'avancée du programme afin de savoir si les nouveaux ajouts ou les modifications effectuées avaient du sens ou si l'on retournait en arrière. À mon sens c'était très utile pour voir l'avancée ainsi que les points qui pouvaient éventuellement bloqués.



Son fonctionnement est assez simple.

Dans un premier temps on va prendre le fichier résultats.CSV

Ensuite on va garder toutes les lignes où le numéro de séquence est égale à 1 afin de ne pas récupérer plusieurs fois la même adresses

On va séparer les adresses retrouvées et les non retrouvées et calculer le taux de correspondance avant de l'afficher

Pour les adresses retrouvées on prends le score de similarités moyen et on forme un histogramme avec et pour les autres on essaie d'extraire les mots les plus fréquents et voir combien de fois ils se répètent afin de chercher un éventuel mot qui apparaîtrait plus souvent et l'ajouter aux mots parasites du programme principal pour qu'ils soient pris en compte quand il faudra faire la normalisation à nouveau.

Donc vous l'aurez compris ce fichier de stats sert essentiellement de test pour être sûr de la qualité du code produits

Figure 15 : Algorithme de statistiques

Au départ le système présenté plus haut a permis d'atteindre un taux de correspondance d'environ 35 % ce qui est quand même peu et montre à quel point les adresses de bases pouvaient être difficile à interpréter. Plus tard, en améliorant le nettoyage des adresses ainsi qu'avec l'inclusion de l'api Geopy qui n'était pas présente au début nous sommes montés à un taux de correspondance supérieur à 74% ce qui représente une énorme amélioration. En ajoutant le fait qu'avec des scores moyens proches entre 0.97 et 1 sur les correspondances retenues, ça montre que le programme est très fiable et les résultats sont encourageants pour une première version faite en si peu de temps.

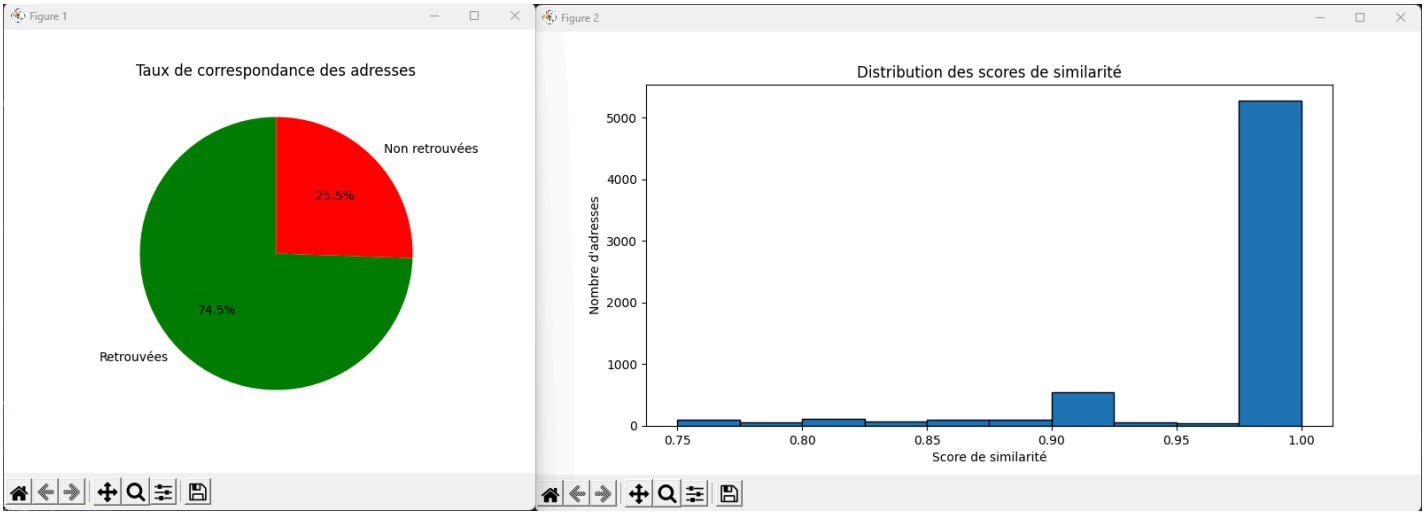


Figure 16 Statistiques

4.5 Limites rencontrées et pistes d'amélioration

Même si le programme de correspondance fonctionne globalement bien, certaines limites ont été observées lors des phases de tests.
Tout d'abord, les adresses TRÈS incomplètes ou mal saisies (par exemple sans numéro, ou avec un nom de voie abrégé ou erroné) restent particulièrement durs à corriger automatiquement. Dans certains cas, même la version nettoyée de l'adresse ne permet pas de trouver une correspondance fiable.

Enfin, la phase de nettoyage dépend énormément de l'enrichissement progressif de la base de mots à exclure. Au fil des tests, de nouveaux cas particuliers apparaissent, ce qui implique une veille continue et des ajustements réguliers dans la logique de nettoyage. C'est ce que j'ai dû faire à chaque nouveau tests afin d'y inclure le plus de mots possibles.

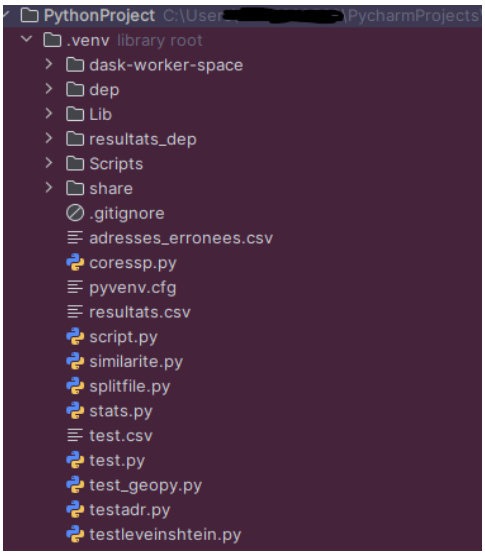
Une piste d'amélioration possible que nous avons envisagé avec un de mes collègues est d'explorer l'utilisation de modèles plus avancés, comme un modèle de machine learning et le modèle de type seq2seq (sequence-to-sequence), capables de corriger une adresse automatiquement en se basant sur un apprentissage préalable. Et le code que j'ai crée pourra servir de base pour le dit apprentissage.

4.6 Réutilisabilité du projet

L'ensemble des programmes, fichiers générés et résultats produits pendant mon stage ont été pensés pour être réutilisables facilement à l'avenir.
Tous les scripts sont séparés en fichiers indépendants et bien organisés : génération des erreurs, normalisation et correspondance, analyse statistique etc…

Le découpage des adresses de la BAN par département facilite leur chargement rapide, et limite les ressources utilisées lors du traitement.
L'utilisation de bibliothèques performantes comme Dask (pour le traitement de gros volumes de données) et Rapidfuzz (pour le calcul rapide de similarité textuelle) garantit de bonnes performances, même sur de grandes bases.

Enfin, le fichier CSV, est structuré contenant toutes les informations nécessaires à une intégration dans une base de données : adresse d'origine, correspondance, coordonnées géographiques, score, etc.



On peut y voir les quelques fichiers que j'ai fait pour chaque utilisations différentes ainsi que des fichiers tests[XXX].py servant d'essaie de nouvelle fonctionnalités avant d'être intégré au code principal
Ce projet pourrait donc très facilement être intégré dans un futur projet comme celui de Horus ou réutilisé dans d'autres contextes.

Figure 17 : Fichier projet

Conclusion

Ce stage à l'URSSAF m'a permis de vivre une expérience à la fois formatrice et technique car il s'agit de ma toute première vraie expérience dans le big data. En partant d'un problème réel (la qualité des adresses) j'ai pu mener un projet complet : de la génération de données erronées à la recherche automatique de correspondances fiables dans une base de données officielle.

Au fil des semaines, j'ai pu approfondir mes compétences en Python, découvrir de nouvelles bibliothèques comme Dask ou Rapidfuzz et d'autres, afin de mieux comprendre les enjeux liés à la fiabilisation des données. J'ai également appris à mieux organiser mon travail, à documenter mes choix, et à proposer des solutions évolutives et réutilisables pour que ce projet soit utilisable par tous.

Ce stage a donc forcément renforcé mon intérêt pour le secteur du Big Data, dans lequel je souhaiterais désormais évoluer pour mes futurs études.
À la suite de mon travail, on m'a d'ailleurs proposé une alternance pour la suite, ce qui m'a énormément encouragé. Cela me permettrait de poursuivre ce même projet sur le long terme, notamment en explorant les pistes d'amélioration évoquées précédemment comme l'intégration d'un modèle de machine learning pouvant corriger automatiquement des adresses pour ne plus avoir à s'inquiéter pour le futur.

Plus globalement, ce stage a été une excellente introduction au domaine du Big Data, et m'a permis de comprendre la quantité énorme de données que l'on reçoit et l'importance de bien les traiter afin de faire fonctionner tous les systèmes d'information.

Le métier de data scientist, par exemple, est un bon exemple : il permet de transformer des données brutes en informations utiles pour aider à la prise de décision ou optimiser des systèmes. Ce type de profil est aujourd'hui recherché dans des environnements très variés, que ce soit à l'URSSAF, mais aussi dans des domaines comme le sport, les jeux vidéo, ou l'environnement.

C'est justement cette diversité qui me motive à continuer dans cette voie, car elle permet de toujours avoir de la variété et donc même si je souhaite changer d'entreprise je serais garantie de travailler sur des sujets tous aussi différents les uns que les autres.

Bilan de compétences

Compétence du BUT Informatique	Mise en pratique durant le stage
Compétence 1 **Réaliser un développement d'application**	Développement en Python : génération d'adresses erronées, nettoyage automatisé, correspondance avec la BAN, export structuré en CSV. Structuration modulaire du code, tests, améliorations et gestion des entrées/sorties de données.
Compétence 2 **Optimiser des applications informatiques**	Utilisation de bibliothèques performantes comme Rapidfuzz pour le calcul de similarité et Dask pour le traitement efficace de gros volumes de données. Optimisation des temps de traitement, gestion mémoire, découpage logique des traitements par département.
Compétence 6 **Travailler dans une équipe informatique**	Collaboration avec mon tuteur, échanges réguliers avec l'équipe sur les choix techniques, autonomie encadrée, adaptation aux outils et aux pratiques internes de l'URSSAF comme Microsoft Teams pour la communication.

GLOSSAIRE

Terme	Définition
API (Application Programming Interface)	Interface permettant à des programmes de communiquer entre eux.
BAN (Base Adresse Nationale)	La base de données officielle des adresses françaises.
Big Data	Traitement de très grands volumes de données complexes et variées.
Dask	Bibliothèque Python pour le traitement parallèle de données volumineuses.
Geopy	Bibliothèque Python pour la géolocalisation d'adresses.
Distance de Levenshtein	Algorithme mesurant les différences entre deux chaînes de caractères.
Machine Learning	Technique d'IA permettant à un programme d'apprendre à partir de données.
Normalisation	Nettoyage et harmonisation des données pour un traitement automatisé.
Rapidfuzz	Bibliothèque Python pour la comparaison rapide de chaînes de caractères.
Regex (Expressions régulières)	Outils permettant de chercher des motifs spécifiques dans du texte.
Score de similarité	Note de 0 à 1 indiquant la ressemblance entre deux textes.
Seq2Seq (Sequence to Sequence)	Modèle d'IA utilisé pour corriger ou traduire automatiquement des séquences de texte.

VPN (Virtual Private Network)	Connexion sécurisée à distance au réseau d'une entreprise.

Bibliographie

Outils techniques

RapidFuzz Documentation. : https://rapidfuzz.github.io/RapidFuzz/Usage/distance/Levenshtein.html

Dask Documentation : https://docs.dask.org/en/stable/

Geopy Documentation : https://geopy.readthedocs.io/

Expressions régulières : https://docs.python.org/3/library/re.html

Matplotlib Documentation : https://matplotlib.org/

Libpostal : https://github.com/openvenues/libpostal

Données institutionnelles et contexte

Base Adresse Nationale : https://adresse.data.gouv.fr

URSSAF : https://www.urssaf.fr

Concepts algorithmiques et IA

Distance de Levenshtein : https://fr.wikipedia.org/wiki/Distance_de_Levenshtein

Sequence to Sequence : https://machinelearningmastery.com/sequence-to-sequence-prediction-neural-networks-python-keras/

ANNEXES

```python
def generer_variantes_adresse(adresse):
    """Génère plusieurs variantes erronées d'une adresse avec des erreurs réalistes."""
    erreur = ["faute","abreviation","suppresion"]
    mots = adresse.lower().split()
    variantes = set()

    for _ in range(10):  # Générer plusieurs variantes de l'adresse
        nouvelle_adresse = []
        for mot in mots:
            nouveau_mot = mot

            if mot.isdigit():
                if random.random() < 0.2:  # 20% de chances de transformer un nombre
                    nouveau_mot = nombre_en_lettres(int(mot))
            else:
                # Définir le nombre max d'erreurs en fonction de la longueur du mot
                if len(mot) <= 3:
                    max_transformations = 1
                elif 4 <= len(mot) <= 6:
                    max_transformations = random.randint( a: 1,  b: 2)
                else:
                    max_transformations = random.randint( a: 1,  b: 3)

                transformations = random.sample(erreur,k=max_transformations)

                for transformation in transformations:
                    if transformation == "faute":
                        if random.random() < 0.6:  # 60% de chances d'appliquer une faute de frappe
                            nouveau_mot = faute_frappe(nouveau_mot)
                    elif transformation == "abreviation":
                        if random.random() < 0.4:  # 40% de chances d'utiliser une abréviation
                            nouveau_mot = abreviation(nouveau_mot)
                    elif transformation == "suppression":
                        if random.random() < 0.5:  # 50% de chances de supprimer une lettre
                            nouveau_mot = supprimer_lettre(nouveau_mot)

            nouvelle_adresse.append(nouveau_mot)
        variantes.add(" ".join(nouvelle_adresse))
    return list(variantes)
```

Annexe 1

Fonction generer_variantes_adresse :
Cette fonction génère des variantes erronées d'une adresse afin de simuler des erreurs courantes pouvant être faites par un humain.
Processus :

L'adresse est convertie en minuscules et splité en mots.

Pour chaque variante (10 au total), chaque mot est modifié selon des règles aléatoires basées sur trois types d'erreurs :

Faute de frappe (faute_frappe) : substitution ou permutation de caractères (60% de chance d'application).

Abréviation (abreviation) : remplacement par une forme abrégée (40% de chance).

Suppression (supprimer_lettre) : suppression d'une lettre aléatoire (50% de chance).

Pour les mots numériques, une conversion en lettres peut être appliquée avec 20% de probabilité (nombre_en_lettres).

Le nombre maximal d'erreurs appliquées dépend de la longueur du mot (de 1 à 3).

La fonction retourne une liste des variantes uniques. Et ce n'est pas afficher dans la capture d'écran mais ensuite ces variantes sont enregistrer dans un CSV

```python
import ...
# Abréviations des types de voies
ABREVIATIONS = {
    "bd": "boulevard",
    "av": "avenue",
    "ch": "chemin",
    "rte": "route",
    "pl": "place",
    "all": "allée",
    "imp": "impasse",
    "st": "saint",
    "ste": "sainte",
    "national": "nationale"
}

def normaliser_adresse(adresse):
    adresse = adresse.lower().strip()
    adresse = re.sub(pattern: r'"', repl: '', adresse)
    adresse = re.sub(pattern: r'\s+', repl: ' ', adresse)
    # Extraire le code postal
    code_postal_match = re.search(pattern: r',\s*(\d{4,5})$', adresse)
    code_postal = code_postal_match.group(1) if code_postal_match else ""
    # Supprimer le code postal temporairement
    adresse = re.sub(pattern: r',\s*\d{4,5}$', repl: '', adresse)
    # Normalisation des accents
    adresse = unicodedata.normalize(form: 'NFD', adresse).encode(encoding: 'ascii', errors: 'ignore').decode("utf-8")
    # Supprimer "CHEZ MME/MR..." et tout ce qui suit jusqu'au premier chiffre
    adresse = re.sub(pattern: r'\b(chez|mme|m\.|mr|mlle|m)\b\s+[^\d]+(?=\d)', repl: '', adresse).strip()
    # Supprimer "CHEZ MME/MR..." en milieu d'adresse
    adresse = re.sub(pattern: r'\b(chez|mme|m\.|mr|mlle|m)\b\s+', repl: '', adresse)
    # Supprimer "ETG X" et "APT X" etc.
    adresse = re.sub(pattern: r'\b(porte|tour|esc|etg|apt|appt|app|appartement|etage|résidence)\s*\d*[a-zA-Z]?\b', repl: '', adresse)
    # Remplacement des abréviations
    for abbr, full in ABREVIATIONS.items():
        adresse = re.sub(pattern: rf'\b{abbr}\b', full, adresse)
    # Nettoyage final des espaces
    adresse = re.sub(pattern: r'\s+', repl: ' ', adresse).strip()
    # Reconstruction de l'adresse avec le code postal
    return f"{adresse},{code_postal}" if code_postal else adresse
```

Annexe 2 : Code de normalisation

Voici le code que j'ai décrits à l'intérieur de mon rapport vous pouvez vous y référer pour mieux comprendre de quoi je parle.