ÉCOLE NATIONALE DES CHARTES
UNIVERSITÉ PARIS, SCIENCES & LETTRES

Jean-Victor BOBY
Ingénieur

La déportation des Roms en Transnistrie, 1942-1944.

Étude de l'appariement de listes de déportés.

Mémoire pour le diplôme de master
« Technologies numériques appliquées à l'histoire »

2022

Résumé

Champ de recherche délimité et relativement récent (fin des années 1990), l'étude de la déportation des Roms roumains déportés par le gouvernement roumain entre 1942 et 1944 dispose néanmoins d'un ensemble de recherches solidement documentées, basées sur un vaste ensemble d'archives témoignant du sort des Roms déportés en Transnistrie. Parmi ces derniers, l'United State Holocaust Memorial Museum (USHMM) dispose d'une collection numérisée [1] comprenant environ 25 000 images de la majeure partie des listes nominatives de Roms visés par les mesures de persécution. Ces listes ont également été océrisées par l'USHMM et traitées avec leurs métadonnées descriptives au format CSV. Ce rapport de stage présente l'analyse des données, leur modélisation ainsi que les outils de traitement développés dans le cadre du projet « La déportation des Roms en Transnistrie, 1942 - 1944. Trajectoires individuelles et destin collectif », qui s'intéresse à une exploitation centrée sur le destin d'environ 25 500 déportés, notamment grâce à une représentation spatiale et temporelle de leur parcours. Un premier volet a consisté à finaliser la modélisation des données et la préparation des livrables auprès du partenaire du projet. Une part importante du travail a consisté à produire une liste de déportés dédoublonnée et de proposer des indicateurs pour en mesurer la qualité, en identifiant les similarités entre individus de différentes listes, afin de reconstituer les trajectoires alors même que les états civils (noms, prénoms, dates de naissance, composition de la famille et rôle familial) sont imprécis et parfois absents. En dernier lieu, nous avons élaboré des stratégies et des outils de traitement pour les futures analyses anticipées, avec un succès relatif.

Mots-clés : Roms ; Déportation ; Recencement ; Rapprochement d'entités ; Mesures de similarité ; Résolution d'entités ; Document océrisé ; Base de données ; XX^{e} s.

Informations bibliographiques : Jean-Victor BOBY, *La déportation des Roms en Transnistrie, 1942-1944. Trajectoires individuelles et destin collectif*, mémoire de master « Technologies numériques appliquées à l'histoire », dir. Grégoire Cousin, Julien Pilla, École nationale des chartes, 2022.

1. RG-25.050M, *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum Archives, Washington, DC., URL : `https://collections.ushmm.org/search/catalog/irn36431` (visité le 16/08/2022).

Remerciements

Je tiens à remercier Grégoire Cousin de m'avoir accueilli au sein de l'École des Hautes Études en Sciences Sociales (EHESS) et m'avoir témoigné sa confiance tout au long de ce stage.

Mes remerciements vont également à Julien Pilla de l'École Nationale des Chartes, directeur de ce mémoire.

Je remercie également Carmen Brando de la plateforme géomatique de l'EHESS pour sa disponibilité et ses précieux conseils ainsi que l'ensemble des équipes de l'EHESS et de KleioLab, partenaire du projet.

Ce mémoire est également le fruit de deux années universitaires passionnantes. Je souhaite ainsi adresser mes remerciements à l'équipe pédagogique et aux enseignants de l'École Nationale des Chartes (ENC) pour m'avoir accueilli au sein du master Technologies Numériques Appliquées à l'Histoire (TNAH), pour la confiance qu'ils m'ont accordée et la qualité de leurs enseignements.

En dernier lieu, je suis infiniment reconnaissant envers ma famille pour leur soutien sans faille, sans lequel je n'aurais pas pu entreprendre ma reconversion professionnelle. Je tiens également à étendre ces remerciements à mes collègues et amis de la promotion TNAH, qui m'ont permis, par leur accueil chaleureux et leur bienveillance, d'aborder un tel projet avec une énergie décuplée.

Table des matières

Introduction

Contexte

L'histoire de la population rom [2] en général et de son génocide durant la Seconde Guerre mondiale reste méconnue du grand public. Ce dernier n'est commémoré à l'échelle européenne que depuis 2015, le 2 août. La recherche historique s'est fortement développée dans les années 1990 grâce notamment à un accès facilité aux archives, à la faveur de la chute des régimes du bloc de l'Est. Parallèlement, la reconnaissance des persécutions passées et des difficultés persistantes des Roms aujourd'hui, si elle reste partielle, s'est accompagnée d'un intérêt croissant d'acteurs publics et d'une meilleure prise en compte de la parole de la communauté rom, en particulier celle des survivants de cet épisode tragique. En ce qui concerne la déportation des Roms en Transnistrie [3], l'historiographie comporte désormais des références significatives [4] et de nombreuses listes de déportés sont accessibles pour la recherche et le grand public [5] mais le champ d'étude reste vaste.

Le projet « La déportation des Roms en Transnistrie, 1942 - 1944. Trajectoires individuelles et destin collectif », financé par la Fondation pour la Mémoire de la Shoah (FMS) [6], a pour objectif d'appréhender les effets de ces persécutions sur le long terme, en proposant un cadre intelligible de la déportation, centré sur le destin des quelques 25 500 déportés. Il entend reconstituer les trajectoires spatiales et sociales des déportés en Transnistrie entre 1942 et 1944 à partir d'une analyse fine des recensements. La spatialisation des parcours et leur visualisation sur la plateforme Geovistory [7], développée par la société Kleiolab [8] (partenaire du projet), devrait permettre de saisir la totalité du phénomène et d'aider à comprendre davantage la composition socio-démographique des populations cibles mais également d'analyser les modalités précises et les logiques internes d'une persécution encore envisagée selon des perspectives trop générales.

2. La plus importante minorité ethnique européenne (estimée entre 10 et 12 millions), voir : *Égalité, inclusion et participation des Roms dans l'UE*, Commission européenne - European Commission, URL : `https://ec.europa.eu/info/policies/justice-and-fundamental-rights/combatting-discrimination/roma-eu/roma-equality-inclusion-and-participation-eu_fr` (visité le 21/08/2022).

3. Voir le rappel historique en préambule.

4. Quelques titres de référence détaillant la déportation des Roms : Viorel Achim, *The Roma in Romanian History*, Budapest, HUNGARY, 2004, URL : `http://ebookcentral.proquest.com/lib/univpsl-ebooks/detail.action?docID=3137208` (visité le 17/08/2022) ; Id., « La déportation des Rroms en Transnistrie, les données principales », *Études Tsiganes*, 56–57–1-2 (2016), p. 66-89, DOI : `10.3917/tsig.056.0066` ; International Commision on the Holocaust in Romania, *et al.* (éd.), *Final Report*, Iași, 2005 ; Ilsen About et Anna Abakunova, *The Genocide and Persecution of Roma and Sinti. Bibliography and Historiographical Review,* Research Report, International Holocaust and Remembrance Alliance, 2016, p. 139, URL : `https://hal.archives-ouvertes.fr/hal-02529522` (visité le 17/08/2022).

5. Voir la base de donnée *Holocaust Survivors and Victims Database* à `https://www.ushmm.org/online/hsv/person_advance_search.php`

6. `https://www.fondationshoah.org/`

7. `https://www.geovistory.com/home`

8. `https://kleiolab.ch/`

Le projet doit ainsi répondre à quatre objectifs majeurs :

— déterminer comment les autorités ont procédé, au cas par cas, pour établir l'ethnicité « *țigan* » [9] à la base de la déportation ;
— identifier les profils socio-historiques détaillés des victimes, à partir des données de structure familiale, les âges et les professions des déportés ;
— dresser une synthèse comparative des camps et ghettos ;
— offrir une vue d'ensemble des parcours des déportés entre 1942 et 1944.

La planification de la déportation et son organisation par les autorités nationales sont largement documentées par l'historiographie. Sur la base des informations recueillies lors d'un recensement secret effectué le 25 mai 1942, la déportation des Roms s'est déroulée en trois phases successives [10], une première concernant les « *țiganii nenomazi* » recencés (Tsiganes nomades) à l'été 1942, étendue par une seconde s'intéressant à certaines catégories de « *țiganii nenenomazi* » (Tsiganes non nomades) et enfin une troisième vague de déportations, préparée à l'automne 1942 pour le printemps 1943 qui n'eut jamais lieu, remplacée par des arrestations et des déportations ponctuelles jusqu'en décembre 1943.

Les objectifs

Afin de suivre ce programme ambitieux, les différentes phases de la déportation sont étudiées séparément par l'équipe de recherche. Lors du stage, le projet s'est concentré sur le recensement de juillet puis la déportation de 13 176 « *țiganii nenomazi* » en septembre 1942.

Le premier enjeu consistait à assister l'équipe dans la production de la liste de déportés dédoublonnée, c'est-à-dire de repérer les similarités entre les individus des différentes listes afin de reconstituer les trajectoires alors même que les états civils (noms, prénoms, dates de naissance, composition de la famille et rôle familial) sont imprécis et parfois absents, en proposant des stratégies d'automatisation de traitement des données, notamment concernant l'élaboration d'indicateurs qualité.

Le second enjeu de la mission était d'assurer la préparation et la mise à disposition de ces données et de leurs métadonnées spatio-temporelles auprès du partenaire du projet Kleiolab (voir la note 8), en vue de leur visualisation sur la plateforme Geovistory (voir la note 7).

9. Dénomination locale en roumain des Roms dans les documents d'archives.
10. Voir le rappel historique en préambule.

Après un bref rappel historique (en préambule) puis la présentation d'un état de l'art relatif au traitement de données historiques à caractère personnel d'une part, et d'autre part, aux mesures de similarité pour le rapprochement d'entités (chapitre 1), nous détaillerons les problématiques liées aux caractéristiques des données de la déportation des Tsiganes « sédentaires » et étudierons l'application concrète de ces méthodes à leur traitement (chapitre 2), pour terminer par la présentation des résultats actuels de leur exploitation (chapitre 3).

Ce rapport est une analyse du travail effectué au cours du stage, à savoir l'exposé des méthodes employées et des outils développés, ainsi qu'une réflexion sur leurs limites et les possibilités de développements futurs.

Les scripts développés lors de ce stage sont, quand à eux, disponibles sur un dépôt GitHub et en annexe du présent rapport [11].

11. Ces documents sont hébergés à l'adresse suivante : `https://github.com/vicpsl/Memoire-TNAH-2022-Boby.git`.

Préambule : rappels historiques

Des origines Rom en Roumanie [12]

L'origine des Roms est longtemps restée mystérieuse et empreinte de fantasmes. Elle comporte aujourd'hui encore de nombreuses zones d'ombre, faute d'avoir laissé des traces suffisamment tangibles avant le XIII^e siècle. Ce sont les linguistes qui les premiers ont su rapprocher la langue *Romani* et le *Sanskrit* à partir du XVIII^e siècle. Sans pouvoir expliciter les causes de la migration Rom depuis l'Inde vers l'Europe, initiée vraisemblablement vers la seconde moitié du premier millénaire, l'étude des influences linguistiques présentes dans la langue ont permis de retracer les grandes lignes de leur parcours : des traces d'arménien, de perse, d'importants emprunts linguistiques au grec puis au vieux-slave témoignent d'une migration par l'Asie Mineure vers l'Empire byzantin (dans l'actuelle Grèce) jusqu'aux Balkans [13].

Des témoignages écrits, parmi les premières mentions européennes des Roms reconnues, émanent des archives de la Valachie et de la Moldavie au XIV^e siècle [14], deux principautés qui forment les parties Est et Sud de la Roumanie actuelle, unifiées au cours des siècles avec, du Nord au centre et à l'Ouest, la partie orientale des Carpates et la Transylvanie, et le delta du Danube ainsi que le Dobrudja sur la mer Noire (voir ci-dessous la figure 1).

12. Sur l'histoire de la Roumanie : Traian Sandu, *Histoire de la Roumanie*, Paris, 2008 ; Catherine Durandin, *Histoire Des Roumains*, Paris, 1995 ; Georges Castellan, *Histoire Du Peuple Roumain*, Crozon, 2002 ; Nicolae Iorga et David Prodan, *A History of Romania : Land, People, Civilization*, Havertown, UNITED STATES, 2019, URL : `http://ebookcentral.proquest.com/lib/univpsl-ebooks/detail.action?docID=6465207` (visité le 21/08/2022).

13. Quelques références détaillant ces aspects : V. Achim, *The Roma in Romanian History…* ; Elena Marushiakova et Vesselin Popov, « Gypsy Slavery in Wallachia and Moldavia. » (25 févr. 2013) ; Angus M. Fraser, *The Gypsies*, 2nd ed, Oxford, UK ; Cambridge, USA, 1995 (The Peoples of Europe) ; M. Hübschmannová, « What Can Sociology Suggest About the Origin of Roms », *Archív Orientální*, 40 (1972), p. 51-64, URL : `https://www.proquest.com/docview/1304094622/citation/5F1D1CF72AE4A49PQ/1` (visité le 19/08/2022) ; Donald Kenrick et Kenrick Donald, *Gypsies, from the Ganges to the Thames*, Hatfield, Hertfordshire, 2004 (Interface Collection, 3) ; D. Kenrick et Gillian Taylor, *Historical Dictionary of the Gypsies (Romanies)*, Lanham, Md, 1998 (European Historical Dictionaries, no. 27) ; Jean-Pierre Liégeois, *Roms et Tsiganes :* 2019 (Repères), DOI : `10.3917/dec.liege.2019.01` ; Yaron Matras, *Romani : A Linguistic Introduction*, Cambridge, 2002 ; J. Andoni Urtizberea, Hanns Lochmuller et Ivailo Tournev, « [Myology and Ethnic Minorities : All Roads Lead to the Roma] », *Medecine sciences : M/S*, 31 Spe 3 (7 nov. 2015), p. 34-38, DOI : `10.1051/medsci/201531s310`, voir également les fiches d'informations sur l'histoire des Roms du Conseil de l'Europe : https ://www.coe.int/fr/web/roma-and-travellers/roma-history-factsheets (visité le 19/08/2022).

14. Des donations d'esclaves en 1385 en Valachie et en 1428 en Moldavie, voir : E. Marushiakova et V. Popov, « Gypsy Slavery in Wallachia and Moldavia. »…, p. 2 ; V. Achim, *The Roma in Romanian History…*, p. 10.

FIGURE 1 – Régions historiques de la Roumanie [15]

Une communauté au plus bas de l'échelle sociale

Dès leur arrivée, les Roms déconcertent : leur provenance est souvent incomprise [16]. Leurs mœurs ne sont pas davantage appréciées et les caricatures vont pénétrer l'imaginaire collectif jusqu'à nos jours. Ils sont rapidement réduits et maintenus en esclavage, parfois même sans personnalité juridique (Valachie et Moldavie). Ailleurs, même lorsqu'ils disposent d'un statut plus libéral, ils font l'objet de persécutions, d'expulsions, de tentatives de sédentarisation forcée et, d'une manière générale, de rejet et discriminations [17].

Néanmoins, les Roms deviennent une main-d'œuvre indispensable et leur présence s'impose dans la société. Avec les Lumières et au XIX^e siècle, l'esclavage est remis en cause, progressivement réformé et aboli de 1783 en Bucovine puis Bessarabie [18] et aboli en 1861 en Roumanie, où la nouvelle constitution de 1918 confère la citoyenneté aux Roms. Dès lors, ils s'organisent (organisations nationales de 1933 à 1941), à l'instar des autres minorités.

15. *Historical Regions of Romania & Neigboroughs*, avec la coll. de Mariusz Paździora et Spiridon Ion Cepleanu, Creative Commons Attribution 3.0, 22 mars 2009, URL : `https://commons.wikimedia.org/wiki/File:RomaniaHistRegions.jpg` (visité le 20/08/2022).

16. Ils sont souvent pris pour des Tatars ou des Égyptiens, d'où les termes « *gypsy* » et « *gitano* ».

17. Voir : E. Marushiakova et V. Popov, « Gypsy Slavery in Wallachia and Moldavia. »… ; V. Achim, *The Roma in Romanian History*…, pp. 13-65 et 69-85 ; Ian F. Hancock, *The Pariah Syndrome : An Account of Gypsy Slavery and Persecution*, 2nd rev. ed., with an index, Ann Arbor, 1987 ; George C. Soulis, « The Gypsies in the Byzantine Empire and the Balkans in the Late Middle Ages », *Dumbarton Oaks Papers*, 15 (1961), p. 141-165, DOI : `10.2307/1291178`, JSTOR : `1291178` ; Sam Beck, « The Origins of Gipsy Slavery in Romania », *Dialectical Anthropology*, 14–1 (1989), p. 53-61, JSTOR : `29790296`.

18. Bucovine et Bessarabie font alors respectivement partie des Empires austro-hongrois et russe.

L'historiographie [19] a montré que depuis leur libération de l'esclavage au milieu du XIX^e siècle, et même pendant l'entre-deux-guerres, il n'y avait pas de politique tsigane à proprement parler en Roumanie. Dans la vie politique et l'opinion en général, il n'y avait pas de « problème tsigane » au sens du « problème juif » invoqué de manière croissante en Europe, y compris en Roumanie. Les Roms étaient plutôt traités comme une catégorie sociale. Aussi ils ne sont pas considérés comme une minorité du ressort de la juridiction du Commissariat général aux minorités (*Comisariatul General al Minoritãþilor*), créé en 1938 [20]. L'attention qui leur est portée comme un groupe ethnique distinct relevait davantage des domaines de la sociologie et de l'ethnographie de la période [21], des recherches qui ont largement contribué à la connaissance de leur situation dans les décennies d'avant-guerre, notamment grâce au recensement de 1930 [22]. Ce dernier recense 262 501 personnes se déclarant d'origine tsigane (1,5 % de la population roumaine). La communauté Rom peut sembler être en passe de s'intégrer de manière croissante au cours de cette décénie [23].

Cependant, l'héritage de plusieurs siècles de persécutions et de préjugés reste prégnant et les Roms restent cantonnés au bas de l'échelle sociale. Le nomadisme, même s'il diminue, continue de représenter un problème pour les autorités, qui n'ont de cesse de l'entraver [24], comme pour une partie de la communauté elle-même [25], qui y voit un frein à une assimilation plus rapide.

Et cette altérité, même si elle est devenue « proche » selon les termes de Viorel Achim [26], sera instrumentalisée par les partisans de l'eugénisme et de la biopolitique émergeant au cours de cette même décennie [27] et sera à l'origine des persécutions à venir.

19. Voir : V. Achim, *The Roma in Romanian History…*, pp. 87-161 ; Marian Viorel Anastasoaie, « Roma/Gypsies in the History of Romania : An Old Challenge for Romanian Historiography », *Romanian Journal of Society and Politics, Vol. 3, no. 1* (, 1er mai 2003), URL : `https://www.academia.edu/2534874/Roma_Gypsies_in_the_History_of_Romania_An_Old_Challenge_for_Romanian_Historiography` (visité le 18/08/2022) ; Petre Matei, « Between Nationalism and Pragmatism : The Roma Movement in Interwar Romania », *Social Inclusion*, 8–2 (2020), p. 305, URL : `https://www.academia.edu/43300578/Between_Nationalism_and_Pragmatism_The_Roma_Movement_in_Interwar_Romania` (visité le 18/08/2022).

20. V. Achim, *The Roma in Romanian History…*, p. 163.

21. Notamment : George Potra, *Contribuţiuni la istoricul ţiganilor din România*, Ed. Fundatia Regele Carol I, 1939) ; Ion Chelcea, *Ţiganii din România monografie etnografică*, Ed. Institutul Central de Statistică, 1944), citées par : M. V. Anastasoaie, « Roma/Gypsies in the History of Romania… ».

22. V. Achim, *The Roma in Romanian History…*, pp. 163–167.

23. *Final Report…*, pp. 223–224.

24. Voir la collection de transcriptions d'archives des autorités : Lucian Nastasă et Andrea Varga, *The Volume of Documents "Gypsies of Romania (1919-1944)"*, 2001, URL : `https://roma-survivors.ro/en/resources/volumul-de-documente-tiganii-din-romania-1919-1944` (visité le 18/08/2022).

25. E. Marushiakova et V. Popov, « 'Letter to Stalin' : Roma Activism vs. Gypsy Nomadism in Central, South-Eastern and Eastern Europe before WWII », *Social Inclusion*, 8–2 (4 juin 2020), p. 265-276, DOI : `10.17645/si.v8i2.2777`.

26. V. Achim, « La déportation des Rroms en Transnistrie, les données principales »…

27. Michael Wedekind, « THE MATHEMATIZATION OF THE HUMAN BEING : ANTHROPOLOGY AND ETHNO-POLITICS IN ROMANIA DURING THE LATE 1930s AND EARLY 1940s », *New Zealand Slavonic Journal*, 44 (2010), p. 27-67, JSTOR : `41759355`.

Le territoire et la population de la Roumanie de 1918 a doublé et la part des minorités (hongroise, allemande, juive, ukrainienne, russe et rom) a triplé pour atteindre 30%, causant craintes pour l'unité de la nation et vives tensions entre communautés [28]. L'État s'engage dans une politique de « romanisation », qui s'accompagne de déplacements de populations d'« étrangers » et de l'octroi de privilèges aux Roumains « de souche ». À l'aube de la seconde guerre, le pays est en proie à une forte instabilité politique [29] et ne peut s'opposer à la perte de territoires, arbitrée par l'Allemagne et ses alliés [30] (figure 2).

Ce contexte permet au maréchal Ion Antonescu [31] d'accéder au pouvoir, où il mène une politique fasciste avec l'appui de l'Allemagne nazie. En août 1941, cette dernière lui confie la Transnistrie, qu'il utilise pour mener à bien son projet d'épuration ethnique [32].

FIGURE 2 – Évolution territoriale de la Roumanie lors du conflit [33]

28. T. Sandu, « La Roumanie, une victoire à la Pyrrhus », *Les cahiers Irice*, 13–1 (2015), p. 155-170, DOI : 10.3917/lci.013.0155 ; *The "Majority Question" in Interwar Romania : Making Majorities from Minorities in a Heterogeneous State*, The Myth of Homogeneity, 22 avr. 2020, URL : https://themythofhomogeneity.org/2020/04/22/the-majority-question-in-interwar-romania-making-majorities-from-minorities-in-a-heterogeneous-state/ (visité le 23/08/2022).

29. Gheorghe Zaharia, « La Vie Politique En Roumanie (1940-1944) », *Revue d'histoire de la Deuxième Guerre mondiale et des conflits contemporains*, 35–140 (1985), p. 53-68, JSTOR : 25729301 ; Philippe Henri Blasen, « De La Nomination Du Cabinet Goga Au Coup d État Du Roi Carol II (28 Décembre 1937 - 10 Février 1938) », *Studia Universitatis Babeș-Bolyai Historia*, 63–2 (2018), p. 111, URL : https://www.academia.edu/38211581/De_la_nomination_du_cabinet_Goga_au_coup_d%CA%BC%C3%89tat_du_roi_Carol_II_28_d%C3%A9cembre_1937_10_f%C3%A9vrier_1938_ (visité le 23/08/2022).

30. Rebecca Haynes, « Germany and the Establishment of the Romanian National Legionary State, September 1940 », *The Slavonic and East European Review*, 77–4 (1999), p. 700-725, JSTOR : 4212960.

31. Surnommé le « Guide » (« *Conducător* »), chef de l'État roumain de septembre 1941 à août 1944.

32. V. Achim, « La déportation des Rroms en Transnistrie, les données principales »… ; Lya Benjamin, « La politique antijuive du régime Antonescu (1940-1944) relative aux juifs de l'ancien royaume et du sud de la Transylvanie », trad. par Andreea Rota, *Revue d'Histoire de la Shoah*, 194–1 (2011), p. 27-62, DOI : 10.3917/rhsho.194.0027.

33. *Romania, 1942*, United States Holocaust Memorial Museum Archives, Washington, DC., URL : https://encyclopedia.ushmm.org/content/en/map/romania-1942 (visité le 19/08/2022).

La déportation des Roms en Transnistrie

Après avoir initié la déportation des juifs en 1941, le gouvernement du maréchal Antonescu décide d'entreprendre celle des Roms vers la Transnistrie[34].

Le gouvernement a d'abord déporté tous les « *țiganii nenomazi* » recencés (Tsiganes nomades), soit 11 441 personnes, à partir du 1er juin 1942 jusqu'au début de l'été 1942.

La déportation a ensuite été étendue en septembre 1942 à 13 176 « *țiganii nenenomazi* » (Tsiganes non nomades) « ayant un casier judiciaire, les récidivistes et ceux qui n'ont aucun moyen de subsistance et qui n'ont pas d'occupation définie leur permettant de subvenir à leurs besoins ».

Une troisième vague de déportations, préparée à l'automne 1942 pour le printemps 1943, fut remplacée par des déportations ponctuelles jusqu'en décembre 1943.

Les conditions déplorables de voyage ou de transport et dans les camps, ainsi que les épidémies de typhus dues à la privation de nourriture et la violence des autorités roumaines, ont entraîné la mort d'environ 11 500 personnes. Après la retraite de l'armée roumaine et le départ des autorités policières du gouvernorat de Transnistrie, la plupart des survivants sont rentrés en Roumanie au printemps 1944.

34. Quelques titres de référence détaillant la déportation des Roms : V. Achim, *The Roma in Romanian History…* ; Id., « La déportation des Rroms en Transnistrie, les données principales »… ; *Final Report…* ; I. About et A. Abakunova, *The Genocide and Persecution of Roma and Sinti. Bibliography and Historiographical Review,…*

Chapitre 1

L'appariement des données : état de l'art

Pour analyser et reconstruire le parcours des Roms déportés, et en particulier pour en proposer une visualisation spatio-temporelle, le projet « *La déportation des Roms en Transnistrie, 1942 - 1944. Trajectoires individuelles, et destin collectif* » doit étudier le déplacement de ces populations sur la base de listes de personnes, établies en divers points des différents trajets. Ainsi il convient de lier, d'apparier le recensement initial des déportés de juillet 1942 avec les listes réalisées à l'embarquement dans les trains et ultérieurement au sein des camps d'internement, afin d'identifier de manière unique les personnes présentes dans plusieurs archives.

Ce couplage de données permettra la consolidation au niveau individuel d'informations complémentaires, de les enrichir et, *in fine*, de disposer de la vue d'ensemble nécessaire pour l'analyse des effets au long cours de ces persécutions.

La tâche est aisée si les enregistrements disposent d'un identifiant unique dans les différentes sources. En revanche, le processus devient difficile en leur absence, en particulier lorsqu'il est confronté à des milliers d'enregistrements et des ensembles disparates.

Dans l'étude de séries de sources, les manques et les incohérences des informations peuvent rendre leur mise en correspondance problématique. C'est d'autant plus vrai dans le cas des documents historiques car l'acquisition des données par des étapes de reconnaissance des caractères imprimés [1] ou, à plus forte raison, d'écriture manuscrite [2], introduit de nouvelles erreurs.

1. Reconnaissance optique des caractères (imprimés) ou *Optical Character Recognition* (OCR)
2. Reconnaissance automatique d'écriture manuscrite ou *Handwritten Text Recognition* (HTR)

1.1 Historique et définition

1.1.1 Définition

Le couplage de données consiste à apparier des enregistrements (noms, dates de naissance, adresses et autres informations) provenant de différentes sources en une seule entité dans un seul fichier.

De nombreux autres termes sont utilisés pour le qualifier comme rapprochement, appariement, fusion (informatique), recoupement (grand public), parfois liage de données, d'enregistrements ou encore de dossiers, de même que l'expression résolution d'entités[3]...

Dans une moindre mesure, cette variété se retrouve en anglais avec *record linkage* (longtemps resté prédominant), *data linkage/linking* ou *data matching* (plus générique), *deduplication*[4]. *Record linkage* est désormais concurrencé par *entity resolution/matching.*

1.1.2 Historique

La pratique des dénombrements et des recensements est multiséculaire[5]. L'intérêt pour une consolidation systématique d'informations issues de telles sources est également ancien et antérieur à l'apparition de l'informatique, notamment pour les sciences faisant appel aux statistiques, comme l'épidémiologie en médecine.

Nous devons l'expression *record linkage* à Halbert L. Dunn[6] en 1946 : « *Each person in the world creates a Book of Life. This Book starts with birth and ends with death. Its pages are made up of the records of the principal events in life. Record linkage is the name given to the process of assembling the pages of this Book into a volume* »[7].

Howard Newcombe[8], un des pères des bases théoriques du couplage d'enregistrements[9], voit déjà en William Farr un précurseur en ce domaine dès le XIX^{e} siècle (il fut directeur de la statistique au *General Register Office*[10].

3. À distinguer de la Reconnaissance d'Entités Nommées (NER) ou *Named Entity Recognition* (recherche d'entités nommées dans un texte et classement d'un point de vue sémantique : personnes, lieux, entreprises, etc.). Toutefois, ces opérations sont parfois associées d'un point de vue technique et les auteurs emploient parfois ces terminologies de manière interchangeable.

4. Par abus de langage : ce terme réfère au dédoublonnage au sein d'un même fichier.

5. Jacques Dupâquier et Michel Dupâquier, *Histoire de La Démographie*, 1985, DOI : 10.3917/perri.dupaq.1985.01.

6. Halbert Louis Dunn (1896-1975), médecin américain, a été l'un des fondateurs de la *National Association for Public Health Statistics and Information Systems* (NAPHSIS) et de l'*Inter-American Statistics Institute* (IASI).

7. Halbert L. Dunn, « Record Linkage », *American Journal of Public Health and the Nations Health*, 36–12 (déc. 1946), p. 1412-1416, pmid : 18016455, URL : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1624512/ (visité le 25/08/2022).

8. Howard B. Newcombe, « Couplage de données pour les études démographiques », *Population*, 24–4 (1969), p. 653-684, DOI : 10.2307/1527541.

9. Howard Borden Newcombe (1914-2005), généticien canadien, fut un pionner du *record linkage*.

10. William Farr (1807-1883), médecin épidémiologiste et statisticien britannique, est nommé en 1837 au *General Register Office* chargé du recensement outre-Manche.

Newcombe mentionne également les travaux en sociologie [11] de Christensen [12] sur la famille aux États-Unis dès les années 30 et diverses autres études.

En 1959, Newcombe contribue à jeter les bases de ce qui allait devenir les fondements des méthodes probabilistes en *record linkage* [13], formalisés de manière scientifique en 1969 par Ivan Fellegi et Alan Sunter [14], une méthode toujours utilisée de nos jours.

Avec l'avènement des techniques d'apprentissage automatique (*Machine Learning*) dans les années 1990, ces méthodes sont affinées et améliorées avec l'introduction de comparaisons approximatives des chaînes de caractères entre des prénoms (Marie et Maria) ou des noms (Nicula et Nicolae) [15], ou des techniques de segmentation des analyses sur les bases volumineuses (comparaisons de personnes du même sexe, du même lieu) [16].

Récemment, les chercheurs ont créé des méthodes de comparaisons uniquement binaires (correspondances ou non-correspondances) basées sur un corpus d'entraînement, ainsi que des méthodes de regroupement (*Clustering*) [17].

Enfin, les derniers développements s'intéressent :

— aux méthodes de graphes [18] (*Graph*) pour caractériser les relations entres les entités afin d'utiliser ces nouvelles données dans le processus de couplage ;

— à l'apprentissage par transfert [19] (*Transfer Learning*), qui permet de transposer des connaissances et des compétences acquises par un système lors de tâches antérieures sur de nouvelles tâches similaires, pour palier au manque de corpus d'entraînement pour ces modèles, pourtant de plus en plus performants ;

— aux problématiques de protection des données personnelles et procédés d'anonymisation [20].

11. Voir : Harold T. Christensen, « I. The Method of Record Linkage Applied to Family Data », *Marriage and Family Living*, 20–1 (1958), p. 38-43, DOI : 10.2307/347362, JSTOR : 347362.

12. Harold Taylor Christensen (1909-2003), professeur à l'université de Purdue (États-Unis), dont il fut le premier directeur du département de sociologie.

13. H. B. Newcombe, J. M. Kennedy, S. J. Axford et A. P. James, « Automatic Linkage of Vital Records », *Science*, 130–3381 (1959), p. 954-959, JSTOR : 1756667.

14. Ivan P. Fellegi et Alan B. Sunter, « A Theory for Record Linkage », *Journal of the American Statistical Association*, 64–328 (1969), p. 1183-1210, DOI : 10.2307/2286061, JSTOR : 2286061.

15. Peter Christen, *Data Matching : Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, Berlin ; New York, 2012 ; Thomas N. Herzog, Fritz Scheuren et William E. Winkler, *Data Quality and Record Linkage Techniques*, New York ; London, 2007.

16. P. Christen, *Data Matching*…

17. David Hand et P. Christen, « A Note on Using the F-measure for Evaluating Record Linkage Algorithms », *Statistics and Computing*, 28–3 (mai 2018), p. 539-547, DOI : 10.1007/s11222-017-9746-6.

18. Nishadi Kirielle, Charini Nanayakkara, P. Christen, Chris Dibben, Lee Williamson, Eilidh Garrett et Clair Manson, « Unsupervised Graph-based Entity Resolution for Accurate and Efficient Family Pedigree Search », dans *Proceedings of the 25th International Conference on Extending Database Technology 2022, Edinburgh*, URL : https://openproceedings.org/2022/conf/edbt/paper-86.pdf (visité le 26/08/2022).

19. N. Kirielle, P. Christen et Thilina Ranbaduge, « TransER : Homogeneous Transfer Learning for Entity Resolution », dans *Proceedings of the 25th International Conference on Extending Database Technology 2022, Edinburgh*, URL : https://openproceedings.org/2022/conf/edbt/paper-27.pdf (visité le 26/08/2022).

20. P. Christen, *Linking Sensitive Data*, 2021, URL : https://link.springer.com/book/10.1007/978-3-030-59706-1 (visité le 26/08/2022).

1.2 Vue d'ensemble/Principes généraux

Nous nous intéressons principalement ici à l'appariement de personnes.

1.2.1 Objectifs et terminologie

Dans le cadre de notre étude, l'objectif de l'appariement entre deux jeux (*dataset*) de données individuelles, parfois nommées micro-données ou données très détaillées (*microdata*) [21], est de déterminer si les paires (*pairs*) de données de deux individus désignent la même personne.

L'ensemble des paires possibles d'un dataset en est le produit cartésien, soit pour deux fichiers de taille n et m, le total des combinaisons est $n \times m$.

Chacun des couples de données peut caractériser une paire d'individus identiques, auquel cas la paire est dite **liée** (*match*), ou une paire d'individus différents, désignée comme **non liée** (*unmatched / no match*). Les deux adjectifs « lié » et « non lié » correspondent aux résultats obtenus par le programme, le modèle ou l'outil utilisé pour qualifier les paires, sans lien avec le statut réel du résultat. Lorsque ce dernier est connu, *i.e.* les données a et b d'une paire réfèrent effectivement au même individu, on parle de **paire annotée**, souvent après un contrôle ou une validation manuelle.

L'appariement s'effectue grâce au degré de similitude entre les données disponibles dans les **variables** ou **attributs** (prénoms, noms, date de naissance, etc.) des disponibles dans les fichiers comparés.

La comparaison peut être **exacte** (*exact matching*) ou **floue** (*approximate string comparison* ou *fuzzy matching*). Dans le premier cas, une correspondance parfaite entre les données des attributs de la paire est requise pour obtenir une paire liée. Cependant, il existe souvent de nombreuses erreurs orthographiques, typographiques ou des variables manquantes dans les fichiers de micro-données individuelles. Il faut alors recourir à la deuxième méthode pour introduire une tolérance plus ou moins importante aux différences présentes dans les données, afin de pouvoir lier des paires qui ne seraient pas parfaitement correspondantes.

Données / Type d'appariement	Fichier 1			Fichier 2			Appariement
	Prénom	Initiale	Nom	Prénom	Initiale	Nom	Résultat
Exact	Ion	N.	Paslea	Ioan		Paslae	Paire non liée
Flou	Ion	N.	Paslea	Ioan		Paslae	Paire liée

TABLE 1.1 – Illustration de couplages exact et flou.

21. Par opposition aux données démographiques qui s'intéressent aux ensembles.

1.2.2 Étapes du processus de couplage

Les principales étapes d'un processus d'appariement et les méthodes les plus communes qui leur sont associées sont les suivantes :

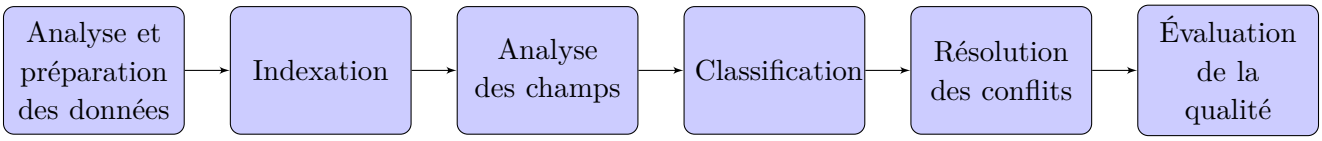


FIGURE 1.1 – Étapes des appariements

1.2.2.1 Standardisation et préparation

Afin de tirer davantage d'information du traitement ultérieur des données, et notamment lors de la comparaison des chaînes de caractères, il convient dans un premier temps de les normaliser et standardiser, en identifiant et en traitant les points suivants.

1. Les données non pertinentes :
   Au niveau des attributs, cela peut consister en la suppression de ceux sans utilité pour l'analyse (texte non structuré et complexe à analyser).
   Concernant les enregistrements, il s'agit notamment de supprimer les caractères spéciaux, la ponctuation, les diacritiques, la capitalisation des noms ou tout autre élément susceptible de causer davantage de bruit (*noise*) qu'il n'apporte d'information.
   Il est également nécessaire de s'assurer que les données pourront être exploitées. Si deux recensements listent l'âge des personnes (3 mois, 41 ans, 1 an, etc.), cette étape permet de transformer ces éléments variables (dans leur forme et dans le temps) par les valeurs fixes et stables comme l'année de naissance (soit, pour un document datant de 1942 : 1942, 1901, 1941, etc.).
2. Les données altérées ou inexactes :
   Des valeurs ou leurs combinaisons peuvent s'avérer inexactes ou contradictoires, par exemple un individu identifié comme un enfant mineur avec un âge de 32 ans ou encore un prénom mal orthographié en « Cnostantin ». Lorsqu'elles sont indéniables, ces erreurs gagneraient à être corrigées, dans la mesure où la valeur correcte peut être déterminée. Préférer une analyse manuelle en excluant de tels enregistrements de l'analyse automatique peut également se révéler judicieux.
3. Les données manquantes :
   De la même manière, il faudra arbitrer entre la suppression ou l'enrichissement en valeurs déduites pour des attributs ou des enregistrements qui seraient insuffisamment renseignés en fonction de l'importance de ces données pour le processus

de couplage. Un exemple serait, sous certaines conditions, d'attribuer un genre masculin manquant à un individu prénommé « Vladimir ». Il sera plus délicat de le faire pour « Valerie ».

4. La segmentation :
   Il peut s'avérer intéressant d'agréger (pour créer des clés de blocage, voir 1.2.2.2) ou de segmenter des informations (adresses avec numéro, nom de voie, code postal et ville) pour les rendre plus exploitables, en fonction des traitements de comparaison envisagés.

   Cependant, extraire les composantes, du nom d'un individu par exemple, peut nécessiter des règles codées complexes. Dans de nombreux cas de figures, ce processus implique la création et la maintenance de nombreuses tables de conversion. Pour palier à ces limites mais également obtenir une plus grande souplesse, des approches statistiques ce sont développées, notamment le modèle de Markov [22] (*Hidden Markov Model*).

5. Les formats incompatibles :
   Pour pouvoir comparer les *dataset* entre eux, le format de leurs attributs doivent être compatibles, notamment en ce qui concerne les chiffres (chaîne de caractères ou nombre), les dates, des coordonnées géographiques, etc.. Pour le genre des individus, un jeu de données disposant d'indicateurs F (féminin) ou M (masculin) ne saurait être directement comparé avec un autre, où les mêmes informations seraient codées « *Female* » et « *Male* ».

L'importance de ces opérations de traitements en amont est nécessaire et souvent indispensable. L'objectif est double : corriger ou supprimer les éléments qui pourraient nuire aux analyses et enrichir des modalités trop peu renseignées, afin de gagner en pertinence sans diminuer la valeur initiale des informations, un équilibre parfois délicat à trouver et qui peut induire la réalisation de différents tests.

1.2.2.2 Indexation et blocage

Nous avons vu que le nombre de paires est égal au carré de la dimension des bases à rapprocher. Pour deux sources de taille moyenne (10 000 enregistrements), cela représente déjà 100 millions de couples possibles à analyser, ce qui est à la fois considérable et peu pertinent. La proportion de paires d'individus uniques s'en trouve par ailleurs fortement diluée, ce qui représente un handicap dans la détermination des paires valides lors de la classification des paires liées et non liées. Le principe de l'indexation est d'écarter en amont les paires indiscutablement incompatibles, du fait de nombreux attributs différents. Les

22. Ou chaîne de Markow : calcul de probabilité de phénomènes aléatoires basé sur le principe d'une interdépendance d'évènements liés les uns aux autres tels les maillons d'une chaîne, utilisé en reconnaissance vocable, l'analyse de séquences biologiques, le traitement automatique du langage naturel, etc.

stratégies de filtrage sont nombreuses, les plus importantes étant le **blocage** (*blocking*) et les algorithmes de « **voisins triés** » (*Sorted Neighbourhood method ou SN*).

Le blocage

Cette méthode, très courante, vise à regrouper à l'avance les couples d'enregistrements les plus susceptibles d'être corrects (*candidate pairs*). Dans l'hypothèse où les entités de deux sources devraient se situer dans la même zone géographique, il est possible d'utiliser le code postal comme **clé de blocage** (*blocking key*), afin de ne comparer que celles qui sont géographiquement proches, diminuant ainsi drastiquement le nombre de comparaisons inutiles et les paires non liées.

Fichier 1 (gauche g)						Fichier 2 (droite d)					
ID	Prénom	Nom	A. Naiss.	Code résidence	Clé	ID	Prénom	Nom	A. Naiss.	Code résidence	Clé
g1	Ion	Paslea	1892	03001	ip03001	d1	I	Paslae	1939	03001	ip03001
g2	Ion	Paslea	1915	03001	ip03001	d2	Ioan	Paslae	1915	03001	ip03001
g3	Ion	Paslea	1892	27003	ip27003	d3	Ion	Paslea	1892	27003	ip27003
g4	Ionita	Paslea		03001	ip03001	d4	Ion	Paslea	1892		ip

TABLE 1.2 – Example fictif de clé de blocage (initales et résidence).

Dans la table 1.2 (où g1 = d4, g2 = d2, g3 = d3, g4 = d1), tous les individus (à l'exception de g3, d3 et d4) sont regroupés par la clé de blocage « ip03001 ».

Cela permet de ne pas comparer l'enregistrement g1 avec d3 car il ne réside pas dans les communes d'intérêt dans notre hypothèse (pourtant, g1 et d3 auraient sans doute été liés). On observe les limites du blocage sur d4 : il ne sera pas comparé à g1 car l'attribut code résidence est manquant, alors qu'il s'agit de la même personne. C'est une des problématiques majeures du blocage : les informations utilisées pour les clés doivent être de bonne qualité et pertinentes (un blocage sur un code géographique aura moins de sens si les personnes ont pu déménager entre les deux sources).

Parmi les méthodes les plus utilisées pour constituer les clés de blocage, citons les codifications phonétiques de champs comme le *Soundex*[23] (prononciations anglo-saxones et ses variantes (*Phonex*, *Phonix*), le *New York State Identification and Intelligence System* (NYSIIS), le *Double Metaphone* (multilingue) ou le *Fuzzy Soundex* qui permettent d'obtenir un code unique à utiliser comme clé couvrant diverses variations orthographiques[24].

23. *Soundex*, FamilySearch Wiki, 24 juin 2022, URL : `https://www.familysearch.org/en/wiki/Soundex` (visité le 30/08/2022).

24. Id., *Data Matching…*, pp. 74-81.

Les approches des voisins triés

Pour cette méthode, élaborée par Hernandez et Stolfo [25], les deux sources sont au préalable concaténées dans un fichier unique, *i.e.* la recherche des couplages consiste alors en une recherche de doublons. Une clé est ensuite calculée à partir d'attributs pertinents pour regrouper les enregistrements par un simple tri effectué sur les clés. Une fenêtre glissante d'une taille prédéfinie balaie le fichier et analyse les références les unes après les autres, limitant ainsi le nombre de comparaisons [26].

Le concept du *SN* a deux principaux inconvénients, d'une part sa grande dépendance à la qualité des attributs choisis au début de la clé car une erreur sur les premiers caractères éloignerait fortement les entrées lors du tri et ils ne seraient pas comparés. L'autre est la taille de fenêtre qui détermine combien de lignes sont comparées les unes aux autres. Si la taille est sous-dimensionnée, des vrais *match* risquent d'être manqués. À l'inverse, avec une dimension trop importante, des rapprochements inutiles seront opérés, affectant potentiellement les performances du modèle. Pour l'améliorer, la réalisation de multiples passages sur des clés différentes ou l'ajustement dynamique de la taille de la fenêtre ont été proposées.

Autres méthodes

D'autres méthodologies de recherche de motifs ont été développées sur ce sujet riche en innovations :

1. l'indexation basée sur les mesures de *n-grammes de caractères* (voir 1.2.2.3) ou de tableaux de suffixes ;
2. l'indexation par *clustering* ou *canopies* ;
3. les techniques d'indexation multidimensionnelle [27].

Indispensable pour les fichiers volumineux, l'étape d'indexation vise à trouver un équilibre entre le coût du nombre de comparaisons à effectuer en termes de capacités computationnelles et de résultats, et la qualité de l'appariement final (une indexation drastique peut exclure des paires valides et créer de faux-négatifs).

25. Mauricio A. Hernández et Salvatore J. Stolfo, « Real-World Data Is Dirty : Data Cleansing and The Merge/Purge Problem », *Data Mining and Knowledge Discovery*, 2–1 (1er janv. 1998), p. 9-37, DOI : 10.1023/A:1009761603038.

26. Voir les illustrations et les explications dans : Patrick Lehti et Peter Fankhauser, « Unsupervised Duplicate Detection Using Sample Non-duplicates », *Lecture Notes in Computer Science*, 4244 (1er janv. 2006), p. 136-164, DOI : 10.1007/11890591_5, pp. 9-11.

27. Zineddine Kouahla, *Indexation dans les espaces métriques Index arborescent et parallélisation*, thèse de doct., Université de Nantes, 2013, URL : https://tel.archives-ouvertes.fr/tel-00912743 (visité le 30/08/2022).

1.2.2.3 Comparaison des champs

Après l'optimisation de la recherche des paires par l'indexation vient la phase d'analyse des couples à apparier. Dans la plupart des cas, un appariement exact ne permettra pas de lier les enregistrements en raison des erreurs et des variations présentes dans des sources qui n'ont souvent, ni été pensées, ni structurées comme de réels dossiers d'identification. Ainsi des outils ont été développés pour effectuer des comparaisons approximatives prenant en compte les nuances induites par des collectes d'information non standardisées. En évaluant le nombre de différences entre deux chaînes de caractères, ils permettent de calculer une mesure de distance et d'estimer le degré de similitude entre les deux données comparées.

Distance de Levenshtein (ou distance d'édition) [28]

C'est la plus commune et elle se définit comme le nombre minimal d'insertions, de suppressions et de remplacements de caractères pour passer d'une chaîne à une autre :

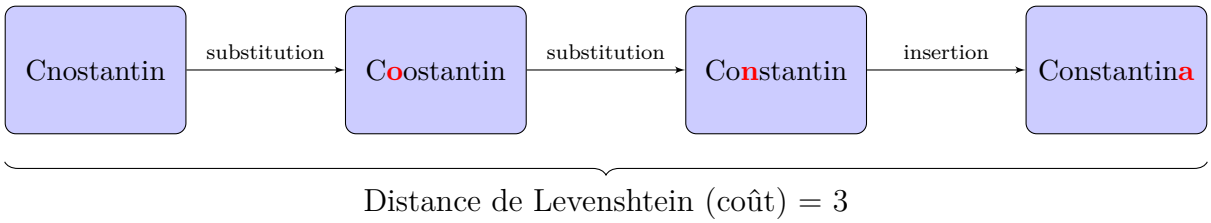


FIGURE 1.2 – Calcul de la distance de Levenshtein.

Pour les appariements, un calcul de similarité est préféré. Il permet d'obtenir un indicateur entre 0 (pas de similarité) et 1 (correspondance exacte). Il est souvent normalisé (noté $sim_{Méthode}$ ci-après), en le rapportant à la longueur de la plus grande chaîne, car une seule altération n'est pas équivalente selon la longueur des mots, soit dans notre exemple :

$$\text{Similarité normalisée } sim_{Levenshtein} = 1 - \frac{distance\,(3)}{longueur\,(Constantina\ :\ 11)} = 0{,}72.$$

La mesure de **Damerau-Levenshtein** [29] est une variante qui prend également en compte la permutation de caractères adjacents. Par conséquent, dans l'exemple ci-dessus, l'inversion des lettres o et n correspond à un changement au lieu de deux. Les distances de Damerau sont toujours inférieures ou égales à celles de Levenshtein (et les similarités plus fortes). Ici, $sim_{Damerau}$ (Cnostantin, Constantina) = 0,82.

28. *Distance de Levenshtein*, Wikipédia, 13 août 2022, URL : `https://fr.wikipedia.org/w/index.php?title=Distance_de_Levenshtein&oldid=196094248` (visité le 31/08/2022).

29. *Distance de Damerau-Levenshtein*, Wikipédia, 17 août 2020, URL : `https://fr.wikipedia.org/w/index.php?title=Distance_de_Damerau-Levenshtein&oldid=173905600` (visité le 31/08/2022).

Similarité de Jaro [30]

Basée sur le nombre et l'ordre des caractères communs entre les deux chaînes de caractères, elle a été notamment développée pour la détection de doublons de noms.

sim_{Jaro} (Cnostantin, Constantina) = 0,936.

Similarité de Jaro-Winkler [31]

Adaptation de la mesure précédente, elle est censée favoriser les chaînes les plus proches sur leurs premiers caractères, en partant du principe que les altérations sont plus fréquemment observées en fin de mot qu'au début.

Ainsi, la position de certains changements sont neutres pour Levenshtein et Jaro :

$sim_{Lev.}$ (C**no**stantin, Constantina) = $sim_{Lev.}$ (Constan**it**n, Constantina) = 0,72 et

sim_{Jaro} (C**no**stantin, Constantina) = sim_{Jaro} (Constan**it**n, Constantina) = 0,936.

En revanche, la similarité Jaro-Winkler pénalise davantage celles situées en début de chaîne, ainsi :

$sim_{JaroWinkler}$ (C**no**stantin, Constantina) = 0,942 obtient un score moindre que :

$sim_{JaroWinkler}$ (Constan**it**n, Constantina) = 0,962.

Smith–Waterman algorithm [32]

Développée à l'origine pour l'analyse de séquences en biologie (étude du génome par exemple), son principal intérêt en data matching est sa relative tolérance aux lacunes, qui rend cette méthodologie intéressante pour l'analyse de groupes de mots comportant des abréviations ou des initiales :

$sim_{Levenshtein}$ (Gheorghe Dumitru Lacatus, Gh D Lacatus) = 0,5 et

$sim_{JaroWinkler}$ (Gheorghe Dumitru Lacatus, Gh D Lacatus) = 0,577,

tandis que :

$sim_{SimthWaterman}$ (Gheorghe Dumitru Lacatus, Gh D Lacatus) = 0,67.

Cette mesure permet en outre d'associer des coûts ou scores particuliers aux différents types de mutations dans les chaînes, selon les besoins (voir la proposition au domaine des appariements par Monge et Elkan).

---

30. *Distance de Jaro-Winkler*, Wikipédia, 22 oct. 2021, URL : `https://fr.wikipedia.org/w/index.php?title=Distance_de_Jaro-Winkler&oldid=187349874` (visité le 31/08/2022).

31. *Ibid.*

32. *Smith–Waterman Algorithm*, Wikipedia, 10 juill. 2022, URL : `https://en.wikipedia.org/w/index.php?title=Smith%E2%80%93Waterman_algorithm&oldid=1097451242` (visité le 31/08/2022).

Ces approches sont relativement efficaces pour apparier en dépit des erreurs orthographiques mais elles sont en revanche très sensibles à l'ordre d'apparition des mots. Des chaînes pouvant être considérées comme liées ou valides obtiennent alors de faibles scores :

$sim_{Levenshtein}$ (Lacatus Gheorghe Dumitru, Gheorghe Dumitru Lacatus) = 0,33,

$sim_{JaroWinkler}$ (Lacatus Gheorghe Dumitru, Gheorghe Dumitru Lacatus) = 0,70,

$sim_{SimthWaterman}$ (Lacatus Gheorghe Dumitru, Gheorghe Dumitru Lacatus) = 0,33.

Similarité basée sur les chaînes atomiques

Pour remédier à ces limitations [33], d'autres méthodes peuvent être mises en œuvre. Ces métriques, dites « n-grammes » (*n-grams*) [34], analysent des sous-ensembles ou « sacs » de n mots (*bags of words*) ou de n caractères alphanumériques (*tokens*) des chaînes. Dans le domaine du couplage, les plus fréquemment citées sont :

Jaccard [35]

La similarité de Jaccard calcule le rapport entre le nombre de n sous-séquences en commun au nombre de toutes les sous-séquences de deux chaînes :

Au niveau des mots, entre « Gheorghe Dumitru Lacatus » et « Gh Dumitru Lacatus », $sim_{Jaccard}$ correspond à 2 (Dumitru et Lacatus en commun) sur 4 (Gheorghe, Gh, Dumitru et Lacatus), soit 0,5.

Elle est peu adaptée aux chaînes courtes [36], par exemple si l'on mesure les bigrammes communs de ioan (io, oa et an) et de ion (io et on) :

- la **distance de Jaccard** est alors :

$dist_{Jaccard}$ (ioan, ion) = 1 - $\dfrac{communs\,(io)}{total\,uniques\,(io, oa, an, on)}$ = 0,75.

- l'**indice de Jaccard** est : $sim_{Jaccard}$ (ioan, ion) = $\dfrac{communs\,(io)}{total\,uniques\,(io, oa, an, on)}$ = 0,25.

33. Les distances d'édition ont d'autres inconvénients, comme l'absence de traitement sémantique, relevant d'autres domaines (traitement automatique des langues) que nous ne détaillerons pas ici.

34. N-*Gram*, Wikipedia, 28 août 2022, URL : `https://en.wikipedia.org/w/index.php?title=N-gram&oldid=1107215398` (visité le 03/09/2022).

35. *Indice et distance de Jaccard*, Wikipédia, 28 mai 2021, URL : `https://fr.wikipedia.org/w/index.php?title=Indice_et_distance_de_Jaccard&oldid=183350601` (visité le 31/08/2022).

36. Il existe des variantes qui peuvent corriger les limites du modèle de base.

TF-IDF (Token Frequency - Inverse Document Frequency) ou Cosinus [37]

Cette méthode est utilisée dans le domaine de recherche d'information, en particulier sur internet. En étudiant parallèlement les fréquences faibles et élevées de termes ou signes dans un corpus, elle permet de dégager ceux qui sont discriminants par contraste avec ceux qui sont très courants. Ainsi, un nom propre de société « Orange », moins commun, aura plus de poids dans le processus d'appariement que le terme « société ».

Ces approches sont efficaces et rapides car elles sont basées sur des calculs relativement simples, notamment le calcul de Jaccard. Elles sont surtout adaptées à l'analyse de chaînes plus longues, comparativement aux mesures de distances, ainsi qu'aux larges corpus (dans le cas du TF-IDF, les poids des termes seront d'autant plus significatifs que le corpus est important). En revanche, les variations réduisent assez fortement leurs capacités de rapprochement.

Mesures hybrides

Les mesures de distances comme les analyses n-grammes ont donc leurs limites. Aussi, pour tenter de combler leurs lacunes respectives, des combinaisons de plusieurs algorithmes ont été proposées pour affiner les modèles.

Monge-Elkan

La mesure de Monge-Elkan a été développée pour comparer de manière floue des chaînes de plusieurs mots (noms de sociétés, noms de personnes non segmentés).

Chaque mot de la chaîne étudiée est comparé à ceux de la deuxième source par un calcul de similarité. En reprenant l'exemple « Gheorghe Dumitru Lacatus » et « Lacatus Gh Dumitru » :

$sim_{JaroWinkler}$ (Gheorghe, Lacatus) = 0
$sim_{JaroWinkler}$ (Gheorghe, Gh) = **0,8**
$sim_{JaroWinkler}$ (Gheorghe, Dumitru) = 0,42
$sim_{JaroWinkler}$ (Dumitru, Lacatus) = 0,52
$sim_{JaroWinkler}$ (Dumitru, Gh) = 0
$sim_{JaroWinkler}$ (Dumitru, Dumitru) = **1**
$sim_{JaroWinkler}$ (Lacatus, Lacatus) = **1**
$sim_{JaroWinkler}$ (Lacatus, Gh) = 0
$sim_{JaroWinkler}$ (Lacatus, Dumitru) = 0,52

37. *TF-IDF*, Wikipédia, 5 janv. 2022, URL : `https://fr.wikipedia.org/w/index.php?title=TF-IDF&oldid=189586794` (visité le 31/08/2022).

Les meilleures combinaisons de paires générées sont donc Gheorges et Gh, Dumitru et Dumitru, Lacatus et Lacatus, dont on fait ensuite la moyenne.

La similarité globale est $sim_{MongeElkan} = \frac{1}{3}(0{,}8 + 1 + 1) = 0{,}93$.

À titre de comparaison, $sim_{JaroWinkler} = 0{,}63$.

Soft-Tf-idf

De la même manière, la méthode TD-IDF est combinée avec une mesure de distance approximative, tolérant les variations. Les fréquences et poids relatifs sont calculés sur un ensemble de mots dont certains sont considérés équivalents, au dessus d'un seuil de tolérance de similarité donné.

Distances phonétiques

Il est également possible d'être confronté à des données dont les variations orthographiques sont trop importantes pour être convenablement traitées par les analyses présentées jusqu'ici.

Editex

Nous évoquions l'intérêt du *Soundex* pour le blocage (voir 1.2.2.2). Le *Soundex* est un code alphanumérique de quatre caractères. Comme pour la distance de Levenshtein, l'*Editex* permet de mesurer et d'attribuer un coût à un changement de prononciation. Il peut donc être pertinent dans l'analyse de chaînes courtes tels que les prénoms ou les noms, notamment pour atténuer les altérations de voyelles, qui ne sont pas codées dans le *Soundex* :

$sim_{Levenshtein}$ (Lixandru, Alexandru) = 0,67,
sim_{Editex} (Lixandru, Alexandru) = 0,83.

Cette propriété peut, à l'inverse, constituer un frein à la discrimination :

$sim_{Levenshtein}$ (Dimitri, Dumitra) = 0,71,
sim_{Editex} (Dimitri, Dumitra) = 0,86.

Syllable Alignment Distance [38]

Le principe de cette méthode consiste à convertir les chaînes de caractères en séquences de syllabes et d'évaluer le nombre d'altérations à effectuer pour les faire correspondre, selon une méthodologie analogue à celle des calculs de distance et de similarité de Smith-Waterman (coûts variables selon les opérations).

38. Voir : P. Christen, *Data Matching…*, p. 118.

Autres méthodes de comparaisons

Parmi les autres approches de comparaison, nous proposons d'aborder brièvement les suivantes :

— Les **plus longues sous-chaînes communes** ou *Longest Common Sub-String* (LCS) [39]. Elles consistent à déterminer la plus longue suite de caractères communs aux chaînes et n'est pas dépendante de l'ordre des mots.

— Méthode des « **co-occurrences** » [40]. Elle cherche à rapprocher des termes que les mesures de distance ne parviendront pas à résoudre (« United States of America » et « USA »), ou à les dissocier (« USA » et « UK », de distance proche) par des associations logiques : « United States of America » et « USA » peuvent être associés car ils sont liés par des références redondantes (« co-occurrence ») à leurs États (Pennsylvanie, Texas, etc.).

— Les **graphes** ou *graph* appliqués à l'appariement [41]. Le concept est ici d'étudier des groupes d'individus ou des personnes dans leurs contextes (membres d'une famille, auteurs et leurs co-auteurs) afin de résoudre le problème de couplage d'homonymes ou de données très éparses et approximatives.

— Comparaison de données numériques (âges, numéros de voie, coordonnées géographiques, etc.) :
Après avoir standardisé les informations (voir les formats), plusieurs approches sont possibles. Ainsi, les éléments peuvent être évalués au format numérique, soit en calculant une différence absolue, soit en mesurant une différence relative en pourcentage entre les valeurs.
Sinon, la mesure d'une distance textuelle sur la chaîne peut se révéler intéressante, notamment en cas d'information trop disparates.
Concernant les adresses, il existe un certain nombre d'outils de codification géographiques et de calcul de distances qui permettront d'évaluer les correspondances [42].

---

39. *Plus longue sous-chaîne commune*, Wikipédia, 31 oct. 2021, URL : `https://fr.wikipedia.org/w/index.php?title=Plus_longue_sous-cha%C3%AEne_commune&oldid=187600673` (visité le 04/09/2022).

40. Rohit Ananthakrishna, Surajit Chaudhuri et Venkatesh Ganti, « Eliminating Fuzzy Duplicates in Data Warehouses » (, 2 août 2002), DOI : `10.1016/B978-155860869-6/50058-5`.

41. *A Graph Matching Method for Historical Census Household Linkage | SpringerLink*, URL : `https://link.springer.com/chapter/10.1007/978-3-319-06608-0_40` (visité le 04/09/2022) ; Byung-Won On, Nick Koudas, Dongwon Lee et Divesh Srivastava, « Group Linkage », dans *2007 IEEE 23rd International Conference on Data Engineering*, Istanbul, 2007, p. 496-505, DOI : `10.1109/ICDE.2007.367895` ; B.W. On, « Social Network Analysis on Name Disambiguation and More », dans 2008, p. 1081-1088, DOI : `10.1109/ICCIT.2008.210`.

42. P. Christen, *Data Matching…*, p. 124 et pp. 208-211.

La diversité des mesures présentées ici témoigne de la variété des cas de figures, des mutations et des irrégularités qui peuvent apparaître dans les données généralement traitées pour le couplage. Elles ont chacune leurs spécificités et leurs limites et correspondent souvent à des usages relativement délimités (efficacité sur des chaînes soit courtes, soit longues), d'où l'intérêt de les combiner dans certains cas. Néanmoins, elles permettent toutes de jauger la similarité des informations entre les attributs de différentes sources, de manière standardisée (scores entre 0 et 1 par exemple). Les données complexes deviennent plus comparables pour les étapes suivantes du processus d'appariement.

1.2.2.4 Comparaison des paires ou « classification » [43]

Dans l'étape de classification, les scores de similarités sont évalués entre les paires candidates (susceptibles d'être réconciliées), afin de les départager entre paires liées, non liées et parfois paires incertaines (paires dont la classification n'a pu aboutir et qui doivent faire l'objet d'un traitement particulier, comme un examen manuel).

Les variables de couplage (*match keys*)

Il convient dans un premier temps de déterminer quels attributs (quelles données) seront utilisés pour réaliser les rapprochements.

Ce choix dépend de la qualité et du pouvoir discriminant des attributs qui seront choisis comme variables de couplage, avec de possibles répercussions sur la qualité de l'appariement.

Comme pour le blocage, les variables de couplage sont des attributs pour lesquels les données sont fiables, relativement bien renseignées et discriminantes, invariables dans le temps (dates de naissance plutôt que l'âge). Cette étape se nourrit de l'analyse des données de la phase de standardisation et de préparation, qui permet en outre de corriger ou d'inférer des informations erronées, altérées ou manquantes, par exemple à l'aide de dictionnaires de références.

La redondance d'information entre les variables est aussi à étudier, notamment pour des modèles d'appariement statistiques (voir ci-dessous) : incorporer des données interdépendantes telles que ville et code postal peuvent affecter le poids relatif de la concordance (ou non concordance) de ces éléments dans l'évaluation des paires. Il est possible de corriger leur importance dans la classification ou de n'en utiliser la seconde que lorsque la première ne se révèle pas adéquate ou déterminante.

Enfin, selon l'approche utilisée, cette étape doit enfin permettre d'évaluer le degré d'importance des variables entre elles car ces pondérations sont nécessaires dans certaines méthodes.

43. Voir : *Ibid.*, pp. 129-162.

Les logiques d'arbitrage dans la classification, en fonction des similarités obtenues, peuvent être extrêmement variées. Elles sont généralement regroupées au sein de deux catégories :

Les méthodes déterministes [44]

Les classifications déterministes (*deterministic* ou *rules-based record linkage*) sont des méthodes « locales », *i.e.* basées sur les éléments descriptifs des entités considérées indépendemment les unes des autres.

Elles sont construites autour d'un ensemble de règles prédéfinies qui déterminent les décisions d'appariement, en fonction des similarités plus ou moins importantes entre les variables de couplage (attributs utilisés pour établir les liens).

Intuitivement, la méthode la plus simple cherche à obtenir une correspondance exacte, par exemple des noms, prénoms, date de naissance et adresse afin d'établir qu'une paire est liée.

Dans la réalité, les informations sont altérées et les règles doivent tenir compte des légères différences. Si les variables contiennent des erreurs, alors des paires liées ne pourront pas être appariées.

Les approches déterministes induisent souvent la recherche d'un équilibre entre un matching parfait (*full exact matching*) et une certaine tolérance aux variations des éléments (*partial exact matching*).

1. Méthode traditionnelle itérative :

   La réconciliation des entités est effectuée en plusieurs étapes successives, utilisant en premier lieu des règles strictes (appariements exacts) puis en assouplissant les critères de correspondance (abandon de certaines variables, introduction de seuils de similarité dégressifs, etc.) sur les éléments les moins discriminants.

   Ces méthodes sont très intuitives et compréhensibles, économes en termes de temps de calcul. De plus, les paires liées à un stade du processus sont généralement exclues des itérations suivantes, limitant l'importance des opérations au fur et à mesure.

   Ce dernier point est l'une des principales limites du modèle : une paire peut avoir été liée lors d'une des premières itérations du processus alors qu'un meilleur résultat aurait obtenu lors des rapprochements suivants.

44. Voir des exemples dans le domaine de la santé : A. Ariel, B.F.M. Bakker, M. de Groot, G. van Grootheest, J. van der Laan, J. Smit et B. Verkerk, *Record Linkage in Health Data : A Simulation Study*, The Hague, 2015.

2. Méthodes basées sur la combinaison des similarités :

   Contrairement à la technique précédente, celle-ci se veut non-itérative et son objectif est de combiner des seuils de similarité pour les variables étudiées, de les pondérer et de les agréger en un score global. En deçà d'un seuil de score global, les paires ne seront pas liées.

   Comme pour le processus itératif, sa mise en place est relativement aisée d'un point de vue technique. Elle est également économe en ressources informatiques.

   Néanmoins les règles sont délicates à déterminer, elles nécessitent une connaissance approfondie du domaine et souvent des essais itératifs pour jauger de leur efficacité à fournir une classification correcte des paires (qui peut être évaluée sur des échantillons dont le statut réel est connu).

   Afin de palier à cette difficulté, des adaptations de l'algorithme de Kuhn-Munkres (optimisation combinatoire des résultats pour obtenir le couplage parfait de poids optimum) ont notamment été proposées [45].

En résumé, les méthodes déterministes sont nombreuses et variées car elles se prêtent bien aux projets d'appariement dédiés à un domaine de recherche particulier par leur flexibilité.

Cependant, elles reposent souvent sur des observations empiriques et le choix des variables, des combinaisons de leurs résultats respectifs et la dégressivité des contraintes sont déterminantes : comment évaluer un degré de similarité suffisant ? [46]

Les méthodes probabilistes ont pour objectif de déterminer de manière plus formelle et mathématique le pouvoir de discrimination des variables de couplage, en estimant la probabilité de leur (non) concordance dans les paires liées ou non liées.

45. ; Sudipto Guha, N. Koudas, Amit Marathe et D. Srivastava, « Merging the Results of Approximate Match Operations », dans *In VLDB*, 2004, p. 636-647 ; *Algorithme hongrois*, Wikipédia, 8 juill. 2022, URL : `https://fr.wikipedia.org/w/index.php?title=Algorithme_hongrois&oldid=195166805` (visité le 05/09/2022).

46. Voir sur ce point la discussion dans : Jiannan Wang, Guoliang Li, Jeffrey Xu Yu et Jianhua Feng, « Entity Matching : How Similar Is Similar », *Proceedings of the VLDB Endowment*, 4–10 (1er juill. 2011), p. 622-633, DOI : `10.14778/2021017.2021020`.

Les méthodes probabilistes (*probabilistic record linkage*) [47]

La plupart d'entre elles sont basées sur les travaux fondateurs de Fellegi et Sunter [48].

Dans ce modèle probabiliste, dit « classique », les paires sont classées entre paires liées, non liées et les paires indécises. L'objectif est de minimiser la proportion de paires indécises, tout en contrôlant les taux d'erreurs dites de « type I » et de « type II » (respectivement taux de faux positifs et de faux négatifs).

Dans ce modèle, le couplage optimal est atteint lorsque le nombre de paires indécises est faible et que les taux d'erreurs, ou de précision et de rappel [49], restent inférieurs aux taux spécifiés par l'opérateur.

L'intérêt de cette approche est qu'elle est applicable dans différents domaines et ne nécessite pas d'entraînement spécifique, n'a pas besoin d'apprentissage supervisé.

Néanmoins, le modèle requiert une estimation des probabilités de (non) concordance dans les paires liées et non liées. Par exemple, la probabilité qu'un mois de naissance corresponde de manière fortuite dans des paires non liées est de 1 mois sur les douze mois de l'année. Les probabilités sont bien plus complexes sur les autres attributs. Il faudra également paramétrer les taux d'erreurs et la nature des mesures de similarité. Enfin, cette approche est basée sur l'hypothèse de l'indépendance entre les variables de couplage.

Dans la réalité, il est fréquent que des attributs soient inter-dépendants (ville, code postal) ou que l'indexation ou la proportion des paires liées dans les données soit déséquilibrée, ce qui peut nuire à la qualité des résultats.

De nombreuses recherches ont été menées pour pallier ces limites, comme la proposition de Winkler [50] d'utiliser une méthode d'estimation des probabilités (algorithme espérance-maximisation [51]) ou les travaux de Jaro [52].

Comme un appariement est une tâche binaire (lié, non lié), les recherches plus récentes se sont orientées vers l'apprentissage automatique (*machine learning*) supervisé et non-supervisé pour classer les paires, ainsi que vers des méthodes de clustering.

47. P. Christen, *Data Matching…*, pp. 133-139.

48. I. P. Fellegi et A. B. Sunter, « A Theory for Record Linkage »…

49. *Précision et rappel*, Wikipédia, 30 mai 2022, URL : `https://fr.wikipedia.org/w/index.php?title=Pr%C3%A9cision_et_rappel&oldid=194125713` (visité le 05/09/2022).

50. William Winkler et William Gov, « Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage », *Journal of the American Statistical Association* (, 18 janv. 2002).

51. *Algorithme espérance-maximisation*, Wikipédia, 1er mars 2022, URL : `https://fr.wikipedia.org/w/index.php?title=Algorithme_esp%C3%A9rance-maximisation&oldid=191518930` (visité le 05/09/2022).

52. Matthew A. Jaro, « Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida », *Journal of the American Statistical Association*, 84–406 (1989), p. 414-420, DOI : 10.2307/2289924, JSTOR : 2289924.

Les méthodes supervisées (*supervised record linkage*) [53]

Ces méthodes permettent de construire des modèles qui apprennent à identifier de manière automatisée les meilleurs règles de classification, à partir d'échantillon de paires annotées, pour lier ou non les paires des datasets à réconcilier.

Les plus communes sont basées sur la régression logistique (*logistic regression*), les *Support Vector Machines* (SVM) [54] et les arbres de décision (*random forest*) [55] comme les *Gradient Boosting Machines* [56].

Leur grand intérêt est leur caractère automatique, cependant la nécessité de fournir des données annotées (le corpus d'entraînement) représente un frein. En effet, et notamment dans le domaine de l'appariement de données personnelles, il est relativement rare de disposer d'échantillons suffisamment adaptés aux cas de figure étudiés.

Par ailleurs, ces modèles sont aussi sensibles à d'éventuels biais qu'il est souvent nécessaire de corriger, le plus souvent par une ou plusieurs phases d'apprentissage actif (*active learning*), où les logiciels et programmes demandent aux opérateurs de classer des paires ambiguës pour affiner les règles de classifications.

Les méthodes non-supervisées (*unsupervised record linkage*) [57]

Pour ne pas avoir à fournir ou construire un corpus d'entraînement, des processus non-supervisés ont été développés.

Ils sont basés sur les méthodes de clustering, en particulier le partitionnement en k-moyennes (*k-means clustering*) [58], le clustering hiérarchique (*hierarchical clustering*) [59] et l'espérance-maximisation (*expectation-maximization*) [60] utilisée par Winkler.

Les méthodes relationnelles ou méthodes globales [61]

Ces techniques exploitent les relations des entités entre elles (liens familiaux) ou les contextes d'intervention (auteurs) pour aider la classification (voir les graphes).

---

53. Robespierre Pita, Samila Sena, Rosemeire Fiaccone, Leila Amorim, Mauricio Barreto, Spiros Denaxas et Marcos Barreto, « Applying Machine Learning to Improve the Accuracy of Probabilistic Linkage », dans 2017 ; P. Christen, *Data Matching…*, pp. :142-147.

54. *Machine à vecteurs de support*, Wikipédia, 24 mars 2022, URL : `https://fr.wikipedia.org/w/index.php?title=Machine_%C3%A0_vecteurs_de_support&oldid=192207500` (visité le 05/09/2022).

55. *Random Forest*, Wikipedia, 30 août 2022, URL : `https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=1107566819` (visité le 05/09/2022).

56. *Apprentissage par renforcement*, Wikipédia, 21 juin 2022, URL : `https://fr.wikipedia.org/w/index.php?title=Apprentissage_par_renforcement&oldid=194726951` (visité le 05/09/2022).

57. Id., *Data Matching…*, pp. 150-153.

58. *K-moyennes*, Wikipédia, 1er mai 2022, URL : `https://fr.wikipedia.org/w/index.php?title=K-moyennes&oldid=193331134` (visité le 05/09/2022).

59. *Regroupement hiérarchique*, Wikipédia, 5 août 2022, URL : `https://fr.wikipedia.org/w/index.php?title=Regroupement_hi%C3%A9rarchique&oldid=195865980` (visité le 05/09/2022).

60. *Algorithme espérance-maximisation…*

61. Id., *Data Matching…*, pp. 154-157.

Au fil du temps et avec les développements informatiques, les méthodologies d'appariement ont évolué de démarches empiriques, vers des modèles statistiques et enfin vers des modèles mobilisant l'apprentissage automatique.

Chacune d'elles se caractérise par ses avantages et inconvénients, de sorte que le choix d'un modèle relève à la fois des caractéristiques intrinsèques des données mais également de la nature du projet (compétence dans le domaine, ressources humaines et techniques, besoin ponctuel ou construction d'une chaîne de traitement pérenne).

La distinction des modèles est d'ailleurs de moins en moins évidente et les recherches s'orientent vers une intégration progressive des techniques vers toujours davantage d'optimisation, que ce soit par l'apprentissage automatique dans la sélection des mesures de similarité ou de blocage ou l'introduction d'active learning dans de nombreux processus.

1.2.2.5 Résolution des conflits

De nombreux systèmes de couplage réconcilient les paires une à une de manière indépendante, ce qui peut mener à des conflits : cela peut se produire si des scores de similarité sont très proches entre plusieurs entités. Un autre cas de figure peut être le suivant : si A est lié à B et que B est lié à C alors que A et C ont été classés comme une paire non liée, il faut résoudre les conflits.

La manière la plus courante de résoudre les conflits fait appel à des algorithmes « gloutons », c'est à dire qui arbitrent de manière locale (au niveau des conflits) sans prendre en compte la qualité globale de l'ensemble des appariements.

Pour une résolution plus globale qui cherche à optimiser la qualité globale des résultats de couplage, il faut faire appel à l'algorithme hongrois évoqué plus haut.

1.2.2.6 Évaluation de la qualité

Une fois le couplage effectué, il convient d'en évaluer la qualité d'une part, afin notamment de pouvoir prendre des mesures correctives s'il s'avérait insatisfaisant ou des pistes d'améliorations.

Un premier indicateur sera le taux d'entités liées qui fournit une première évaluation globale l'efficacité de l'appariement.

Si cette mesure peut être réalisée dans tous les cas, elle ne permet pas de juger de la pertinence des résultats.

Pour évaluer la qualité du couplage[62], il est essentiel de savoir si les liens obtenus font en fait référence à la même entité, c'est-à-dire s'il s'agit de vrais liens. Si de vrais liens sont connus, le nombre de faux positifs (faux liens) et de faux négatifs (faux non-liens) peut être calculé, selon la « matrice de confusion » :

		Statut réel	
		Entités identiques (P)	Entités différentes (N)
Statut prédit	Lié (PP)	Vrais positifs (VP)	Faux positifs (FP)
	Non lié (PN)	Faux négatifs (FN)	Vrais négatifs (VN)

TABLE 1.3 – matrice de confusion.

Les indicateurs de qualité sont mesurés comme suit[63] :
La qualité mesure les prédictions correctes sur l'ensemble des cas.

$$\text{Qualité} = \frac{\textit{nombre de paires correctement classées}}{\textit{nombre total de paires}} = \frac{VP + VN}{VP + VN + FP + FN}$$

La précision (*Precision*) évalue la part des prédictions correctes sur l'ensemble des prédictions positives, soit le pourcentage de réponses correctes, *i.e.* si le modèle lie un seul cas juste, la précision est de 1. Cet indicateur à lui seul n'est pas suffisant.

$$\text{Précision} = \frac{\textit{nombre de paires liées correctement}}{\textit{nombre de paires liées par le modèle}} = \frac{VP}{VP + FP}$$

Le rappel (ou *Recall*) quantifie les prédictions correctes sur l'ensemble des paires véritablement liées.

$$\text{Rappel} = \frac{\textit{nombre de paires liées correctement}}{\textit{nombre d'entités identiques}} = \frac{VP}{VP + FN}$$

Précision et rappel doivent être analysés ensemble. Si l'objectif est de limiter le nombre de paires liées de manière erronée, il faut privilégier une meilleure précision, le modèle étant alors davantage sélectif. S'il s'agit de détecter plus de paires potentiellement correspondantes, l'accent doit porter sur un meilleur rappel.

Enfin le F1-score (ou *F1-measure*) est largement utilisé pour comparer les modèles entre eux (sur un même problème). Il correspond à la moyenne harmonique de la précision et du rappel, entre 0 et 1 (plus il est élevé, meilleurs sont les résultats). À titre indicatif, un F1-score de 50% signifie que le modèle fait deux erreurs pour un vrai positif.

$$\text{F1-score} = 2 \text{ x } \frac{\textit{Précision} \text{ x } \textit{Rappel}}{\textit{Précision} + \textit{Rappel}} = \frac{2\ VP}{2\ VP + FP + FN}$$

62. *Ibid.*, pp. 163-184.

63. *Sensitivity and Specificity*, Wikipedia, 22 août 2022, URL : `https://en.wikipedia.org/w/index.php?title=Sensitivity_and_specificity&oldid=1105990559` (visité le 05/09/2022).

En résumé, une meilleure précision peut se traduire par un faible rappel et inversement. L'indicateur le plus pertinent doit être favorisé aux regards des attentes, du niveau de risque accepté. De nombreux modèles permettent de régler ces paramètres afin que les résultats soient plus adaptés à la finalité de l'appariement, intimement liée à ces enjeux.

1.3 Enjeux

Les enjeux sont nombreux et majeurs, tant pour le secteur public que le secteur privé.

Pour le secteur public, la médecine, en particulier l'épidémiologie, a été parmi les premiers secteurs pour lesquels il était crucial de rapprocher des informations, tant pour réaliser des études que pour assurer le suivi des patients. Nombre de projets et publications [64] relevant du *record linkage* émanent de ce secteur [65].

En matière de sécurité nationale et de police, le coupage des données devient primordial dans les processus de vérification d'identité et nécessaire dans le traitement, le suivi des affaires et leurs évaluations [66]. C'est également vrai en prévention et détection de la fraude, comme pour l'identification des bénéficiaires effectifs d'avoirs fonciers et financiers.

De même, la statistique publique a largement recours à ces technologies, avec pour enjeu la gestion administrative d'une part, et d'autre part les études statistiques et le suivi des politiques publiques dans de nombreux domaines [67].

Concernant les entreprises, les enjeux sont tout autant significatifs, à l'heure du développement du commerce mondialisé et de l'e-commerce. Dans des sphères très compétitives, connaître ses clients et fournisseurs est essentiel, nécessitant à la fois le nettoyage et la réconciliation, l'enrichissement et la protection des bases de données.

Parallèlement, face à la profusion de données, notamment personnelles, les réglementations deviennent plus contraignantes et de nouvelles bonnes pratiques se développent.

64. Merran Smith et Felicity Flack, « Data Linkage in Australia : The First 50 Years », *International Journal of Environmental Research and Public Health*, 18–21 (28 oct. 2021), p. 11339, DOI : 10.3390/ijerph182111339, pmid : 34769852 ; Adrian Sayers, Yoav Ben-Shlomo, Ashley W Blom et Fiona Steele, « Probabilistic Record Linkage », *International Journal of Epidemiology*, 45–3 (1er juin 2016), p. 954-964, DOI : 10.1093/ije/dyv322 ; N. Kirielle, C. Nanayakkara, P. Christen, *et al.*, « Unsupervised Graph-based Entity Resolution for Accurate and Efficient Family Pedigree Search »…, Quelques exemples :

65. Notons que l'un des centres de recherches nationaux de santé publique en Angleterre dédié au *data linkage* a été nommé le Farr Institute, d'après William Farr, pionnier du domaine (voir l'historique).

66. Jack Short et Brian Caulfield, « Record Linkage for Road Traffic Injuries in Ireland Using Police Hospital and Injury Claims Data », *Journal of Safety Research*, 58 (1er sept. 2016), p. 1-14, DOI : 10.1016/j.jsr.2016.05.002 ; Mauricio Sadinle et Stephen E. Fienberg, « A Generalized Fellegi–Sunter Framework for Multiple Record Linkage With Application to Homicide Record Systems », *Journal of the American Statistical Association*, 108–502 (2013), p. 385-397, JSTOR : 24246450.

67. *Un Outil d'appariement Sur Identifiants Indirects — Courrier Des Statistiques N6 - 2021 | Insee*, URL : https://www.insee.fr/fr/information/5398689?sommaire=5398695 (visité le 05/09/2022).

1.4 Les appariements et la recherche

Avec la révolution informatique puis numérique des dernières décennies, des sources d'information plus exhaustives, plus diverses mais aussi plus dispersées sont devenues disponibles. La littérature scientifique témoigne de l'intérêt croissant des chercheurs pour l'exploitation de ces ressources, désormais permise par les progrès réalisés dans le traitement des données volumineuses, la modélisation et le traitement automatique.

Ces sources, prises séparément, ont toutes leurs limites, que ce soient les données de panel susceptibles d'être biaisées ou les données administratives, fiables et disponibles sur le temps long, mais pauvres en informations relatives aux sujets de recherche.

Le croisement de ces données a ouvert et continuera de renforcer la validité et la pertinence des travaux, notamment pour les études longitudinales. Le développement des techniques de couplage ont accompagné ces efforts, notamment aux États-Unis, au Canada, en Australie ou au Royaume-Uni.

En France, qui dispose pourtant d'une information administrative riche, les acteurs constatent un retard relatif, notamment dû aux réglementations de protection des données en vigueur jusqu'à une époque récente[68]. Pour palier cette situation, l'accès des données a été facilité par les réformes récentes et la mise en place du réseau Quertelet[69], du Centre d'Accès Sécurisé Aux Données (CASD)[70] et de la plateforme des données de santé (« Health Data Hub »)[71], notamment pour l'accès aux données sensibles.

Concernant les archives historiques, signalons plusieurs projets récents ou en cours :

- le projet SOCFACE[72], de grande ampleur, qui vise à numériser analyser les recensements de 1836 à 1936 pour étudier les changement sociaux sur un siècle ;
- un autre projet ANR « Qui est devenu un Nazi ? », dont l'objectif est de créer une base de données des questionnaires de dénazification des zones d'occupation française et américaine en Allemagne (DeNazDB)[73] ;
- le croisement du fichier central à d'autres sources, une expérimentation aux Archives Nationales[74].

68. Lois Informatique et Libertés (1978) et sur l'obligation, la coordination et le secret en matière de statistiques (1951), avant les changement de 2004, 2008 et la loi pour une république numérique (2016) : Nathalie Picard et Kamel Gadouche, *L'accès Aux Données Très Détaillées Pour La Recherche Scientifique*, 2017-06, THEMA (THéorie Economique, Modélisation et Applications), Université de Cergy-Pontoise, 2017, URL : `https://ideas.repec.org/p/ema/worpap/2017-06.html` (visité le 06/09/2022).

69. `https://data.progedo.fr/` du TGIR Progedo `https://www.progedo.fr/`

70. `https://www.casd.eu/`

71. `https://www.health-data-hub.fr/`

72. Projet ANR-21-CE38-0013, `https://anr.fr/Projet-ANR-21-CE38-0013`

73. Projet ANR-21-FRAL-0005, `https://anr.fr/Projet-ANR-21-FRAL-0005`

74. *Fichier central. Archives nationales, carnet de recherche.* URL : `https://labarchiv.hypotheses.org/tag/fichier-central` (visité le 06/09/2022).

En effet, grâce aux progrès des techniques de numérisation et d'acquisition des informations des archives, de nombreux corpus sont désormais accessibles à la recherche [75].

Enfin dans des domaines légèrement différents, l'appariement géographique soutient le développement des référentiels géohistoriques tels que le *Dictionnaire topographique de la France* [76], Paris Time Machine [77], ainsi que la consolidation de référentiels d'autorités en bibliothéconomie, comme les projets SudocAD [78], repris par le projet ANR Qualinca [79].

75. Voir aussi le projet POPP de numérisation des recensements parisiens de l'entre-deux guerre : *POPP. Projet d'océrisation des recensements parisiens*, URL : `https://popp.hypotheses.org/` (visité le 06/09/2022).

76. `https://dicotopo.cths.fr/`

77. `https://paris-timemachine.huma-num.fr/`

78. Michel Chein, Michel Leclère et Yann Nicolas, « SudocAD : A Knowledge-Based System for the Author Linkage Problem », dans *Knowledge and Systems Engineering*, dir. Van Nam Huynh, Thierry Denoeux, Dang Hung Tran, Anh Cuong Le et Son Bao Pham, Cham, 2014 (Advances in Intelligent Systems and Computing), p. 65-83, DOI : 10.1007/978-3-319-02741-8_8.

79. Projet ANR-12-CORD-0012, `https://anr.fr/Projet-ANR-12-CORD-0012~` : *Objectifs | QUALINCA*, URL : `https://www.lirmm.fr/qualinca/index8b01.html?q=fr/objectifs` (visité le 06/09/2022).

Chapitre 2

Étude de l'appariement
des archives de la déportation des Roms

Il sera question dans ce chapitre d'évaluer la faisabilité, le type de solutions à élaborer et leur rentabilité au regard des objectifs de la mission, en prenant en compte d'une part les caractéristiques des données sources, mais également le contexte organisationnel du projet. En outre, on discutera de la sélection des outils déployés pour répondre aux besoins.

2.1 Analyse des données

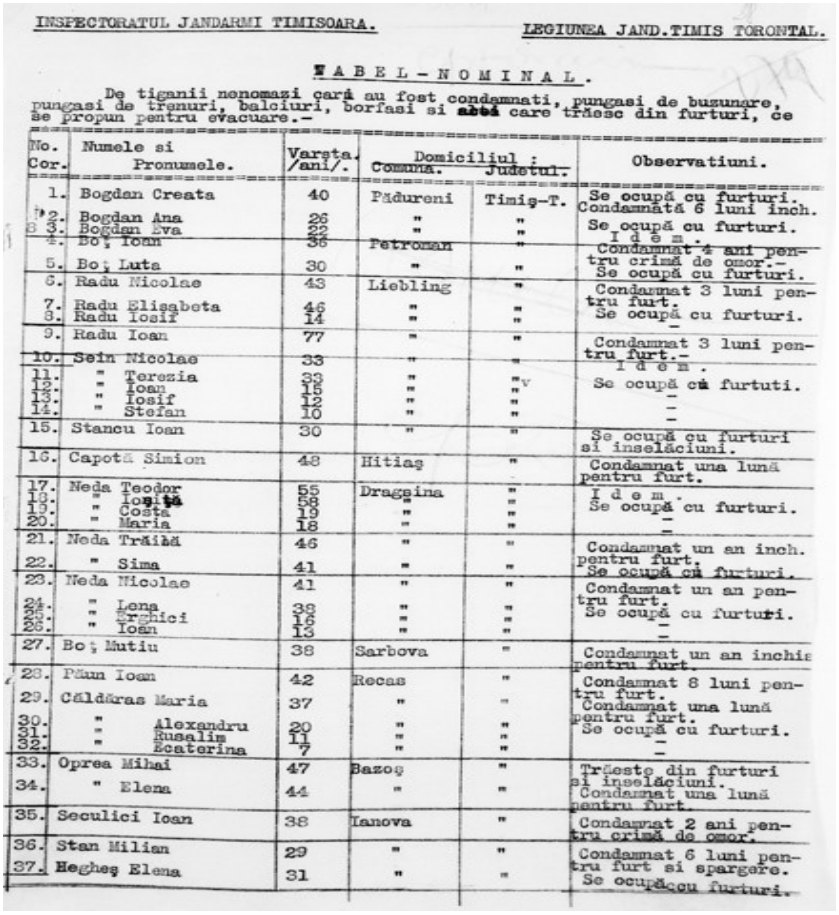
2.1.1 Présentation générale des sources

La collection numérisée [1] comporte environ 25 000 images issues, entre autres, des extraits des fonds des inspectorats généraux de gendarmerie et police, couvrant la majeure partie des listes nominatives de 25 000 Roms visés par les mesures de persécution et de déportation.

L'USHMM a reçu la collection des Archives nationales roumaines (*Arhivele Naţionale ale României*) au cours du projet d'archives internationales de l'USHMM en décembre 2008 (microfilms 1-60) puis en mai 2010 (microfilms 61-64).

Il est important de noter que les listes sont produites à l'échelle locale : départements, villes et exceptionnellement inspectorats régionaux de gendarmerie ou de police. L'analyse portera donc sur ces évènements, département par département, pour les 62 départements contrôlés par le gouvernement Roumain en 1942 (en excluant la Transnistrie).

1. RG-25.050M, *Selected Records from Various Archives of Romania Concerning Roma…*



INSPECTORATUL JANDARMI TIMISOARA. LEGIUNEA JAND.TIMIS TORONTAL.

TABEL-NOMINAL.

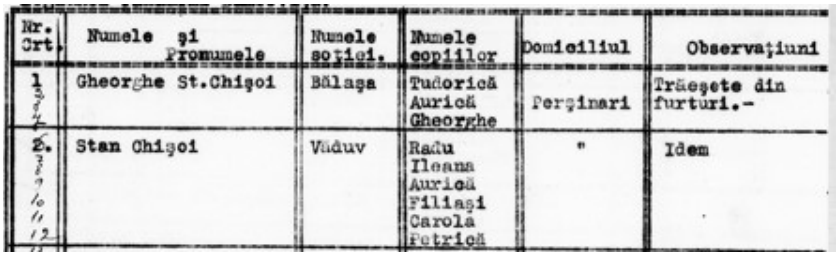
De tiganii nomazi cari au fost condamnati, pungasi de buzunare, pungasi de trenuri, balciuri, borfasi si alti care trăesc din furturi, ce se propun pentru evacuare.-

No. Cor.	Numele si Pronumele.	Varsta /ani/.	Domiciliul : Comuna.	Judetul.	Observatiuni.
1.	Bogdan Creata	40	Pădureni	Timiş-T.	Se ocupă cu furturi. Condamnată 6 luni inch.
2.	Bogdan Ana	26	"	"	Se ocupă cu furturi.
3.	Bogdan Eva	22	"	"	I d e m .
4.	Boţ Ioan	36	Petroman	"	Condamnat 4 ani pentru crimă de omor.-
5.	Boţ Luta	30	"	"	Se ocupă cu furturi.
6.	Radu Nicolae	43	Liebling	"	Condamnat 3 luni pentru furt.
7.	Radu Elisabeta	46	"	"	Se ocupă cu furturi.
8.	Radu Iosif	14	"	"	-
9.	Radu Ioan	77	"	"	Condamnat 3 luni pentru furt.-
10.	Sein Nicolae	33	"	"	I d e m .
11.	" Terezia	33	"	"	Se ocupă cu furtuti.
12.	" Ioan	15	"	"	-
13.	" Iosif	12	"	"	-
14.	" Stefan	10	"	"	-
15.	Stancu Ioan	30	"	"	Se ocupă cu furturi si inselăciuni.
16.	Capotă Simion	48	Hitiaş	"	Condamnat una lună pentru furt.
17.	Neda Teodor	55	Dragsina	"	I d e m .
18.	" Ioşifa	58	"	"	Se ocupă cu furturi.
19.	" Costa	19	"	"	-
20.	" Maria	18	"	"	-
21.	Neda Trăiaă	46	"	"	Condamnat un an inch. pentru furt.
22.	" Sima	41	"	"	Se ocupă cu furturi.
23.	Neda Nicolae	41	"	"	Condamnat un an pentru furt.
24.	" Lena	38	"	"	Se ocupă cu furturi.
25.	" Urghici	16	"	"	-
26.	" Ioan	13	"	"	-
27.	Boţ Mutiu	38	Sarbova	"	Condamnat un an inchis pentru furt.
28.	Păun Ioan	42	Recas	"	Condamnat 8 luni pentru furt.
29.	Căldăras Maria	37	"	"	Condamnat una lună pentru furt.
30.	" Alexandru	20	"	"	Se ocupă cu furturi.
31.	" Rusalin	11	"	"	-
32.	" Ecaterina	7	"	"	-
33.	Oprea Mihai	47	Bazoş	"	Trăeşte din furturi si inselăciuni.
34.	" Elena	44	"	"	Condamnat una lună pentru furt.
35.	Seculici Ioan	38	Ianova	"	Condamnat 2 ani pentru crimă de omor.
36.	Stan Milian	29	"	"	Condamnat 6 luni pentru furt si spargere.
37.	Heghes Elena	31	"	"	Se ocupă cu furturi.

FIGURE 2.1 – Exemple de liste nominative de proposition à la déportation [2].

2.1.2 Des documents de qualité variable

Les listes ayant été établies par de très nombreux acteurs, leurs formes et leurs contenus sont très hétérogènes :



Nr. Crt.	Numele şi Pronumele	Numele soţiei.	Numele copiilor	Domiciliul	Observaţiuni
1	Gheorghe St.Chişoi	Bălaşa	Tudorică Aurică Gheorghe	Perşinari	Trăeşte din furturi.-
2.	Stan Chişoi	Văduv	Radu Ileana Aurică Filiaşi Carola Petrică	"	Idem

FIGURE 2.2 – Liste par famille sur plusieurs colonnes, sans âges [3].

2. Liste de l'inspection de la gendarmerie de Timișoara : RG-25.050M (File ID : 45931), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=45931` (visité le 07/09/2022), Copyright *Arhivele Naţionale ale României*.

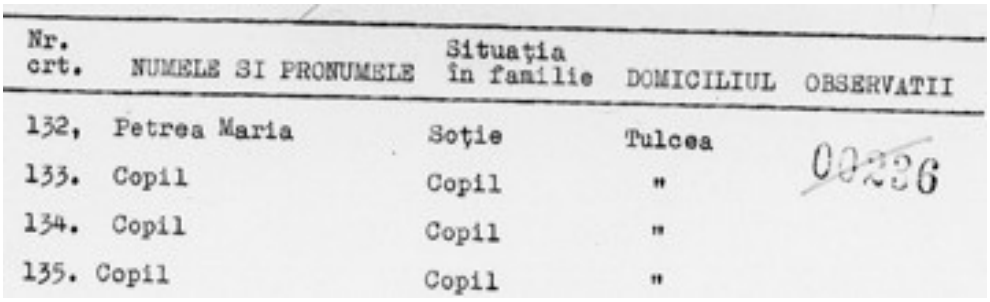
3. RG-25.050M (File ID : 45964), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=45964` (visité le 08/09/2022), Copyright *Arhivele Naţionale ale României*.

T A B L O U.-

De tiganii fără ocupatiune precisă,din care să trăiască în mod cinstit si cari urmează să fie internati în lagăr.-

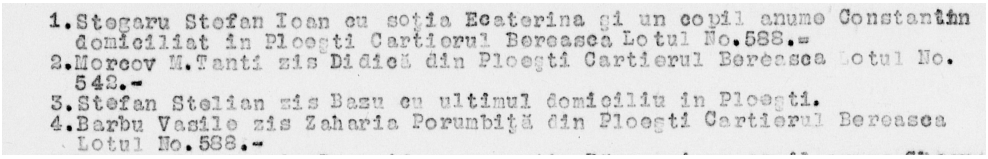
Nr. crt.	NUMELE SI PRENUMELE	A D R E S A.	OBSERVATII
1.-	Manole Tănase c	Dăicăreanu	Nr.12
2.-	Mutu Dumitru c	Florica	Nr.17
3.-	Anita Radu f	Armistitiului	56
4.-	Alexandru Radu b	"	56
5.-	Floarea Radu f	"	56
6.-	Gheorghina Ion Dumitru f	"	56.-

FIGURE 2.3 – Liste sans indicateur de famille, sans âges [4].



Nr. crt.	NUMELE SI PRONUMELE	Situaţia în familie	DOMICILIUL	OBSERVATII
132.	Petrea Maria	Soţie	Tulcea	03236
133.	Copil	Copil	"	
134.	Copil	Copil	"	
135.	Copil	Copil	"	

FIGURE 2.4 – Liste avec des noms manquants [5], sans âges [6].



1.Stegaru Stefan Ioan cu soţia Ecaterina şi un copil anume Constantin domiciliat în Ploeşti Cartierul Berească Lotul No.588.-
2.Morcov M.Tanti zis Didică din Ploeşti Cartierul Berească Lotul No. 548.-
3.Ştefan Stelian zis Basu cu ultimul domiciliu în Ploeşti.
4.Barbu Vasile zis Zaharia Porumbiţă din Ploeşti Cartierul Berească Lotul No.588.-

FIGURE 2.5 – Liste non tabulaire, sans âges [7].

4. RG-25.050M (File ID : 46036), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=46036` (visité le 08/09/2022), Copyright *Arhivele Naţionale ale României*.

5. *Copil*, qui signifie enfant, est utilisé dans la colonne *Numele si pronumele* (Noms et Prénoms).

6. RG-25.050M (File ID : 46415), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=46415` (visité le 08/09/2022), Copyright *Arhivele Naţionale ale României*.

7. RG-25.050M (File ID : 46521), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=46521` (visité le 08/09/2022), Copyright *Arhivele Naţionale ale României*.

La masse de données disponibles (125 000 entrées individuelles pour environ 1 500 documents) a induit, pour le chercheur, une analyse et une réconciliation des sources.

Celle-ci a consisté à suivre les logiques et évènements administratifs afin de trier le corpus. L'USHMM ayant inventorié et numéroté les différents dossiers d'archives, l'exploration de la documentation et de l'historiographie a permis d'en dégager les grandes lignes et identifier 1 083 documents nominatifs.

Les documents couvrent divers aspects de ces événements, notamment [8] :

— le recensement secret (mai 1942) ;
— la déportation des Roms « nomades » (juin - septembre 1942) ;
— les listes des Roms « sédentaires » sélectionnés pour la déportation (juillet 1942) ;
— les listes des Roms « sédentaires » déportés (septembre 1942) ;
— les documents de transfert des Roms « sédentaires » (septembre 1942) ;
— les évasions (1942-1944) ;
— le recensement en préparation de la seconde déportation (octobre 1942) ;
— le rapatriement des Roms mobilisés [9], déportés par erreur (1942-1943) ;
— les listes des déportés dans les camps (1942-1943) ;
— les déportations ponctuelles (1943) ;
— les listes de rapatriés (avril - mai 1944) ;
— des courriers de réclamation des déportés pour un rapatriement (1942-1944).

Cette phase du projet s'intéresse en particulier au rapprochement des listes soulignées ci-dessus, soit la pré-sélection des *țiganii* « sédentaires » de septembre 1942 et les départs effectifs à la fin du même mois, ainsi que l'organisation des trajets (voir la section 3.2.1.2), afin que le prestataire puisse réaliser un prototype de cartographie temporelle.

En effet, le recensement de septembre fournit les adresses de départ, les listes à l'embarquement confirment la liste des personnes déportées, tandis que les documents de transport détaillent la répartition dans les trains et les trajets de ces derniers.

De plus, le recensement et la déportation des Tsiganes « sédentaires » constituent les épisodes pour lesquels les sources sont à la fois les plus exhaustives et les plus importantes.

Concrètement, le recensement de juillet concerne 18 000 personnes et la déportation de septembre 13 176 personnes, selon les documents du ministère de l'intérieur de 1942.

Ces listes ont été transcrites par l'USHMM et ses partenaires, puis préparées au format CSV, avec certaines métadonnées descriptives.

8. Ainsi que divers autres documents sans liens directs avec la déportation.
9. En effet, certains Roms sont incorporés dans les forces armées du régime. La déportation concerne d'abord sur ceux qui ne l'étaient pas.

2.1.3 Le fichier CSV

Par la suite, le recensement de septembre sera désigné régulièrement sous le terme de « Census » et les données de la déportation sous celui de « Déportation » [10].

2.1.3.1 Le fichier global

Toutes les données de la première déportation sont fournies en un fichier CSV unique (133 000 lignes environ), dont la structure est la suivante :

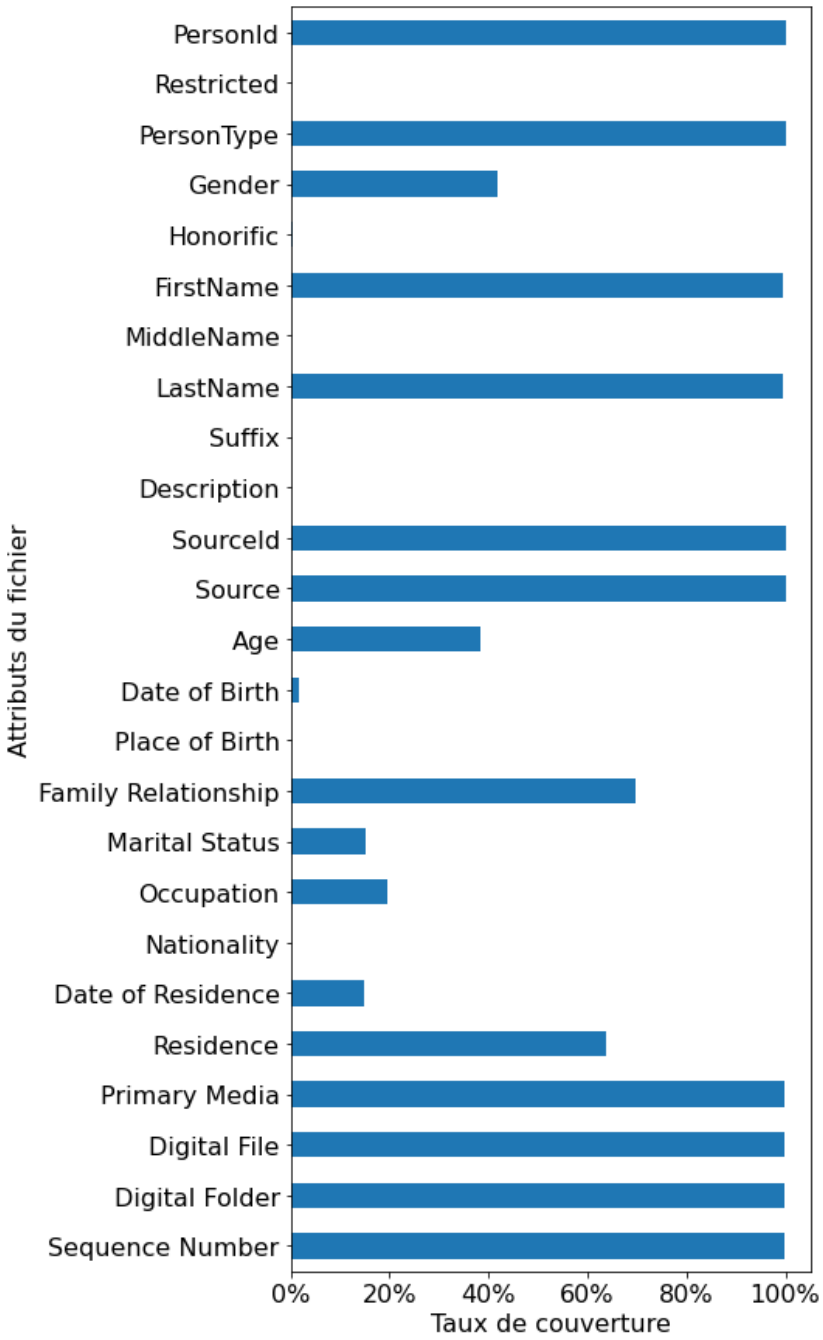


FIGURE 2.6 – Structure de la source CSV (voir le code).

10. Par convenance et pour une meilleure analogie avec les noms des fichiers (census et deportation).

Après un examen détaillé, les attributs suivants ont être écartés :

— champs vides : Restricted, MiddleName, Suffix, Description, Place of birth

— champs sans apport quantitatif ou qualitatif : Honorific, Nationality

— métadonnées sans objet pour l'étude : PersonType (« *Victim* » pour toutes les entrées), Primary Media (redondance avec Digital File).

2.1.3.2 Les données relatives à la première déportation

L'analyse des dossiers d'archives avait permis d'identifier que 317 d'entre eux portaient sur ces évènements, pour respectivement 21 549 personnes recensées en septembre et 13 033 répertoriées dans les listes de déportation [11]. L'échantillon concerné par l'étude a par ailleurs le même taux de renseignement que le fichier complet :

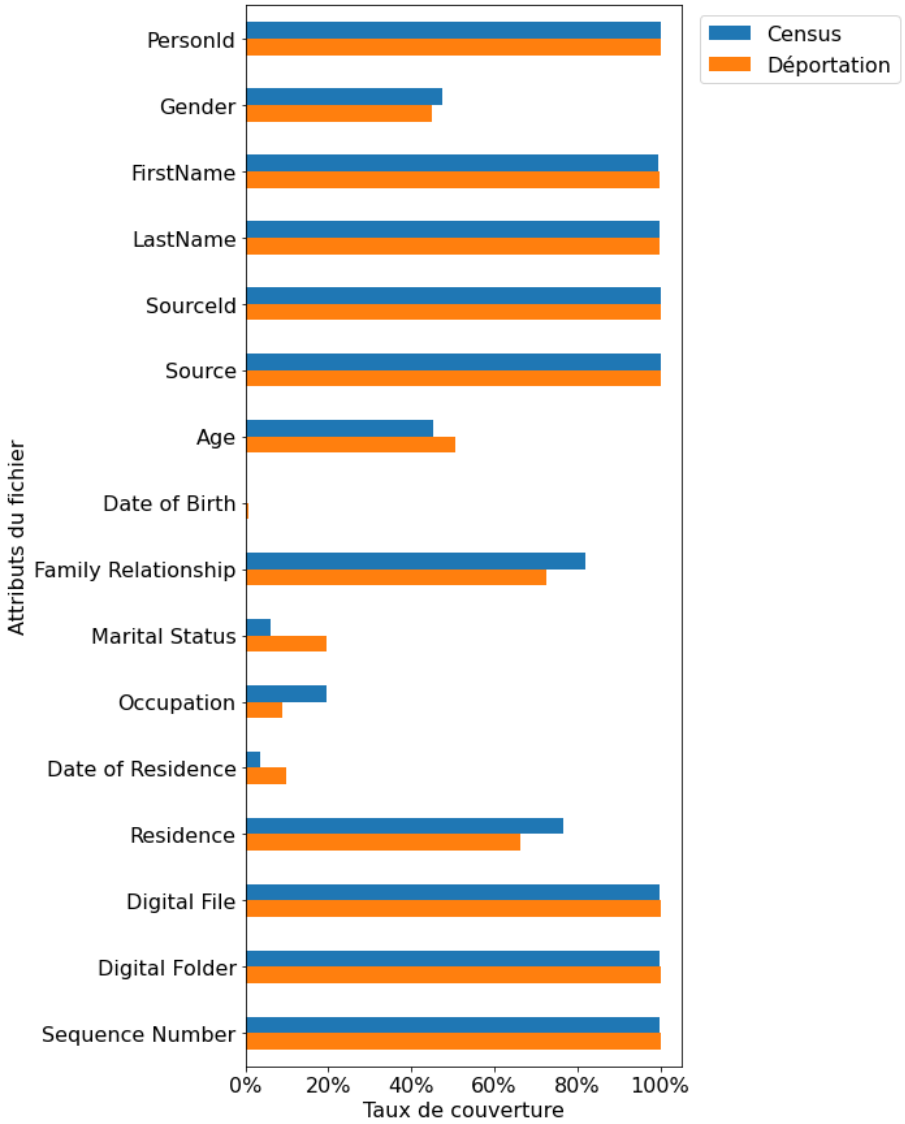


FIGURE 2.7 – Structure confirmée sur l'échantillon de l'étude (voir le code).

11. Sur la base du seul CSV, d'autres ont pu être identifiées et ajoutées manuellement : toutes les sources n'étaient pas incluses et disponibles dans le fichier CSV.

2.1.4 Analyse des attributs disponibles

De fortes disparités de taux de couverture des attributs sont observés dans les fichiers, une analyse est nécessaire pour identifier ceux qui peuvent être utilisés pour l'étude, y compris pour évaluer la qualité des champs convenablement fournis en informations. En outre, lorsque des champs sont susceptibles de se compléter, ils sont étudiés ensemble.

2.1.4.1 Les données originales

Gender (Genre)
Indicateur du genre des personnes, dont la détermination à partir les archives n'est pas toujours aisée, ce qui se reflète dans le taux de disponibilité insuffisant :

— sur environ un quart des documents le genre est convenablement précisé ;
— dans la moitié des documents, seul le genre des adultes est précisé soit via une colonne genre où les hommes sont étiquetés avec la lettre « B » et les femmes avec la lettre « F », soit via le statut marital de mari et épouse ;
— le genre des enfants, étiquetés avec la lettre « C », n'est qu'exceptionnellement précisé. Comme pour les adultes, le statut familial peut clarifier « fils » ou « fille ».

Gender	# Census	# Déportation
Female	7617	4014
Male	2610	1834
NaN [12]	11322	7185

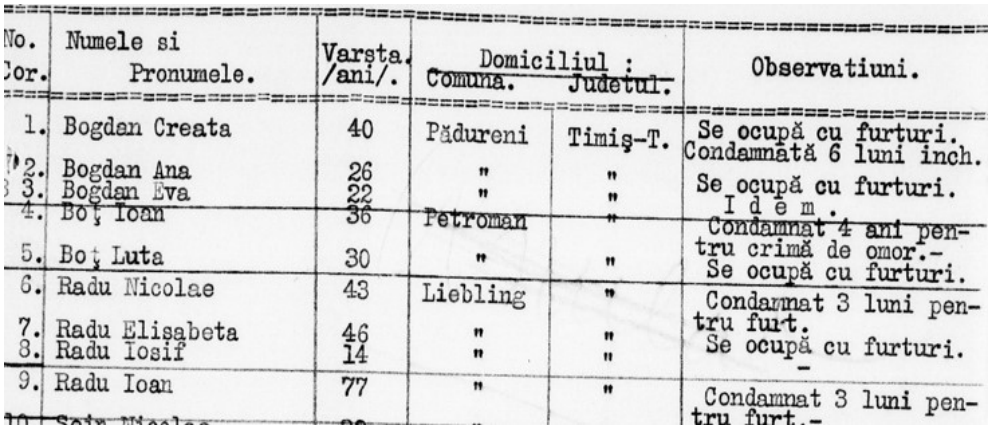
TABLE 2.1 – Disponibilité du genre.

L'absence de genre sur des entrées qui laissent pourtant peu de place au doute devrait permettre d'imputer certaines valeurs manquantes pour enrichir cet attribut.

Family Relationship (Relation au chef de famille)

Indicateur reflétant la position des individus par rapport au chef de famille (homme ou femme), ce dernier étant noté « *Self* » (les célibataires peuvent également recevoir ce marqueur). En effet, lorsque les groupes familiaux sont décrits dans les listes (voir l'exemple suivant 2.8, mais également les figures 2.2 et 2.5), le CSV reprend ces informations comme ci-dessous :

12. « NaN » signifie champs nul.



No. Cor.	Numele si Pronumele.	Varsta /ani/.	Domiciliul : Comuna.	Judetul.	Observatiuni.
1.	Bogdan Creata	40	Pădureni	Timiş-T.	Se ocupă cu furturi. Condamnată 6 luni inch.
2.	Bogdan Ana	26	"	"	Se ocupă cu furturi.
3.	Bogdan Eva	22	"	"	I d e m .
4.	Boţ Ioan	36	Petroman	"	Condamnat 4 ani pentru crimă de omor.-
5.	Boţ Luta	30	"	"	Se ocupă cu furturi.
6.	Radu Nicolae	43	Liebling	"	Condamnat 3 luni pentru furt.
7.	Radu Elisabeta	46	"	"	Se ocupă cu furturi.
8.	Radu Iosif	14	"	"	-
9.	Radu Ioan	77	"	"	Condamnat 3 luni pentru furt.-

FIGURE 2.8 – Detail de liste [13].

	PersonId	Gender [14]	First Name	Last Name	Age	Family Relationship	Marital Status	Residence	Sequence Number	
	8768310		Creata	Bogdan	40	Self		Pădureni, Timiş-T.	1000	
+1 [15]	8768311		Ana	Bogdan	26			Pădureni, Timiş-T.	1010	+10 [16]
+1	8768312		Eva	Bogdan	22			Pădureni, Timiş-T.	1020	+10
+1	8768313		Ioan	Boţ	36	[17]		Pădureni, Timiş-T. [18]	1030	+10
+1	8768314		Luta	Boţ	30			Pădureni, Timiş-T. [18]	1040	+10
+1	8768315		Nicolae	Radu	43	Self		Liebling, Timiş-T.	1050	+10
+1	8768316		Elisabeta	Radu	46			Liebling, Timiş-T.	1060	+10
+1	8768317		Iosif	Radu	14			Liebling, Timiş-T.	1070	+10
+1	8768318		Ioan	Radu	77	Self		Liebling, Timiş-T.	1080	+10

TABLE 2.2 – La transcription CSV du document en figure 2.8

13. Liste de l'inspection de la gendarmerie de Timișoara : RG-25.050M (File ID : 45931), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=45931` (visité le 07/09/2022), Copyright *Arhivele Naţionale ale României*.

14. Ici, le genre n'est pas renseigné ni par le code de lettres « B », « F », « C », ni par le statut familial (uniquement « Self », neutre).

15. Incrémentation unitaire pour chaque personne, de haut en bas, de gauche à droite le cas échéant.

16. Incrémentation par dix pour chaque personne, de haut en bas, de gauche à droite le cas échéant.

17. Ici, « Self » manquant.

18. La transcription est erronée : la ville aurait dû être Petroman.

Les valeurs possibles pour le statut familial sont :

Valeur	# Census	# Déportation
Self	6283	3301
Child	6123	3547
NaN	3902	3578
Wife	2671	1048
Daughter	1844	1048
Relative	171	53
Concubine	166	83
Spouse	156	156
Husband	87	0
Mother	57	40
Brother	20	32
Sister	19	12
Relative-in-law	17	23
Other	0	22
Grandchild	16	41
Other	5	12
Relative-in-Law	4	7
Granddaughter	4	32
Son	0	7
Father	2	3
Sibling	0	3
Grandmother	1	0
Grandfather	1	1
relative	0	2
Aunt	0	1

TABLE 2.3 – Distribution des entrées de statut familial.

La forte disparité, avec des entrées anecdotiques (« Aunt ») et des redondances (comme « Spouse » et « Wife ») induiront une standardisation de cet attribut, en particulier si l'on envisage de l'utiliser dans des règles d'appariement.

Marital Status (Statut marital)

Champs complémentaire des deux précédents, il décrit le statut marital des intéressés, avec trois valeurs possibles uniquement : « Married » - marié(e), « Widowed » - veuf(ve) - et « Single » - célibataire.

Les deux tableaux suivants analysent l'interdépendance de ces trois derniers attributs :

Census			
Gender	Family Relationship	Marital Status	Nombre
Female	Wife	NaN	2276
Female	Daughter	NaN	1830
Female	NaN	NaN	1124
Female	Child	NaN	1054
Female	Wife	Married	393
Female	Self	NaN	391
Male	Self	NaN	1957
NaN	Child	NaN	4893
NaN	Self	NaN	3206
NaN	NaN	NaN	2525

TABLE 2.4 – Correspondance de trois attributs (Census) [19]

Déportation			
Gender	Family Relationship	Marital Status	Nombre
Female	Daughter	NaN	1046
Female	Wife	Married	905
Female	NaN	NaN	522
Female	Child	NaN	374
Male	Self	Married	789
Male	Self	NaN	454
Male	Child	NaN	281
NaN	Child	NaN	2886
NaN	NaN	NaN	2705
NaN	Self	NaN	1175

TABLE 2.5 – Correspondance de trois attributs (Déportation) [20]

Il ressort de l'analyse que très peu de valeurs :

— se contredisent ;

— permettraient de déduire le genre à partir des deux autres variables, étant que la plupart des genres non renseignés sont associés à statuts familiaux neutres, de type « Self » ou « Child » ;

confirmant que le genre devra être déduit des prénoms, là où cela est possible.

FirstName (Prénom)

Prénom ou prénoms (*pronumele*) des personnes. Le champs « MiddleName » n'ayant pas été utilisé, certaines entrées sont relativement complexes. Ces entrées doivent être analysées avec le nom, notamment parce qu'il est nécessaire de tenir compte des spécificités de la formation des noms en Roumanie.

19. Les dix plus importantes quantités, voir l'analyse complète en annexe A.4.
20. Id., voir l'analyse complète en annexe au tableau A.2.

LastName (Nom de famille, Patronyme)

Nom(s) de famille (*numele*) des personnes. Comme pour les prénoms, plusieurs caractéristiques influent sur les données retenues et répertoriées.

Note sur les données personnelles dans le contexte du projet

Au moment où ces données sont produites, l'identité civile et l'identification restent largement floues en Roumanie. Ainsi, de nombreuses personnes peuvent être désignées par leur prénom et leur véritable patronyme mais aussi par l'association de leur prénom avec le prénom du parent (en général du père), voire les deux.

En outre, il convient de souligner que certaines dénominations sont parfaitement interchangeables, avec des prénoms mixtes et d'autres très utilisés en tant que patronymes et inversement.

En résumé, nous retrouvons ici l'éventail des difficultés évoquées par Peter Christen [21] pour ce type de données :

— variantes orthographiques, qu'elles soient issues de variantes réelles dans les graphies possibles (*Ioan* et *Ion*) ou d'erreurs (*Anexandru* pour *Alexandru*) ;
— variations phonétiques, résultant sans doute d'un enregistrement effectué oralement, avec des prononciations proches mais des orthographes relativement distinctes ;
— des appellations alternatives, notamment des diminutifs (*Veta* pour *Elisabeta*) ou des surnoms (*Alexandru zis Romica* pour « Alexandru, dit Romica ») ;
— l'emploi d'initiales, relativement fréquentes dans le corpus.

Dans la documentation de la déportation des Roms, et en dehors des nuances culturelles indiquées plus haut, peuvent s'ajouter les écueils suivants :

— l'agrégation de divers prénoms et noms ;
— l'ordre prénom - nom inversé ;
— les abréviations des informations, causées notamment par l'espace restreint induit par le caractère tabulaire de l'enregistrement (*Ctin* et *Constantin*) ;
— les erreurs liées au mauvais placement des signes de répétition dans les tableaux ;
— les ratures et reprises.

Ces déformations sont exacerbées par l'état de conservation de certains documents, ainsi que par l'étape de transcription, qui a introduit de nouvelles variations ou confusions.

21. P. Christen, « A Comparison of Personal Name Matching : Techniques and Practical Issues », dans *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, 2006, p. 290-294, DOI : 10.1109/ICDMW.2006.2.

Age (Âge)

L'âge des individus, généralement en chiffres, à l'exception des enfants de moins de un an, noté : « 1/12 [1] » pour 1 mois. Par ailleurs, les documents ne datant pas tous de la même année, cette information est variable dans le temps, ce qui nécessitera une standardisation.

Sur la donnée elle-même, les rapprochements préliminaires révèlent de possibles approximations (de l'ordre de plus ou moins trois à cinq ans), plus importantes pour les personnes âgées (parfois jusqu'à dix ans).

Date of Birth (Date de naissance)

L'année de naissance des personnes (4 chiffres), très peu renseignée (50 sur le périmètre de l'étude). Une variable qui ne pourra pas être utilisée pour le rapprochement en l'état, mais qui peut être enrichie à partir de celle des âges.

Occupation (Emploi)

Emploi ou catégorie d'activité, ce champs relativement peu complété fournit des informations très variables sur le statut des personnes, de la profession à la situation militaire (mobilisé ou non) ou judiciaire (condamnations), etc.. En dehors des notes concernant la mobilisation qui permet en partie de reconstituer les sélections de populations réalisées en 1942, cette variable est difficilement exploitable pour l'appariement, mais elle peut être intéressante pour le volet ethnographique de la recherche.

Residence (Adresse)

Les adresses des personnes enregistrées, souvent limitées au lieu-dit ou village, avec parfois des adresses complètes en ville, le département (*județ*) étant souvent mentionné.

Cette donnée est affectée par des variations orthographiques du même ordre que les noms et prénoms, dont nous livrons un exemple ci-dessous.

Cette variable a d'ailleurs fait l'objet d'une standardisation et un identifiant alphanumérique a été attribué aux différents lieux. Elle a également été enrichie lorsque cela était possible, avec les informations présentes au niveau des dossiers. C'est cette dernière donnée que nous utiliserons par la suite.

Adresse
Tăndărei, Ialomiţa
Tăndărei, Ialomiţa || Galaţi
Tămdărei-Ialomiţa
Toudarei, Ialomita
Taudarei, Ialomita
Tăuderei, Ialomita
Tandarei, Ialomița
Tanderei, Ialomița
Tandarei, Ialomiţa
Tăndarei, Ialomița
Tanadrei, Ialomița
Tândărei, Ialomița
Tăndărei, Ialomiță || Tăndărei, Ialomiță
Tăndărei
Tăndărei, Salomița
Tandărei
Tandărei || Galaţi
Tăndarei
Tândarei
Tandarei
Tănlărei
Tăniărei
Ţăndărei
Ţăndărei, Ialomiţa
Ţăndărei, Ialomiţ
Ţăndărei, Ialomita

TABLE 2.6 – Différentes graphies d'un lieu de résidence.

2.1.4.2 Les métadonnées

En ce qui concerne l'appariement, les deux variables les plus importantes sont les deux suivantes :

PersonId (Identifiant individuel)

Numéro de 7 chiffres attribué aux personnes (voir l'exemple au tableau 2.2), « unique » si ces dernières ne sont pas mentionnées à différentes reprises dans une ou plusieurs sources.

Cet identifiant permettra de constituer les identifiants des personnes et de les lier dans les bases de données, une fois l'appariement réalisé.

Sequence number (Numéro de séquence)

Un code numérique (4 chiffres) identifiant l'ordre d'apparition d'une personne dans une liste, en principe de haut en bas pour les lignes, de gauche à droite à l'intérieur d'une ligne, d'un bloc d'information. La première personne reçoit le chiffre 1000, qui est incrémenté de dix en dix le long de la liste (voir la table 2.2).

Cette information peut s'avérer utile au rapprochement. En effet, les deux événements, recensement et déportation, sont proches dans le temps. Contrairement aux couplages de recensements distants de dizaine(s) d'années, la composition des foyers ou leurs adresses ne devraient pas avoir beaucoup évolué.

Date of Residence (Date de résidence)

La date à laquelle l'adresse de résidence est enregistrée, peu fourni, il apporte peu d'informations qui ne sont pas déjà connues à partir de la date des documents eux-mêmes.

Il pourrait en revanche servir d'indicateur lorsqu'il est associé à des adresses précédentes des individus [22].

Digital File (Fichier numérique)

Le nom du fichier numérique d'une archive numérisée, à raison de un par page (« rg-25_050m_0013_00000539 »).

Digital Folder (Répertoire numérique)

Le nom du répertoire ou dossier dans lequel sont stockés les fichiers numériques (« RG-25_050M_0013 »).

Source (Nom de la source)

Un champs mentionnant le nom original de la source (*TABLOU DE TIGANI MOBILIZABILI*) ou attribué par l'USHMM (*[Documents related to persecuted minorities]*).

SourceId (Identifiant de la source)

Le numéro (5 chiffres) du document d'archive dans la collection de l'USHMM [23].

22. C'est le cas sur d'autres documents, notamment ceux des camps.

23. La fiche descriptive de la source est obtenue par une recherche à l'URL `https://www.ushmm.org/online/hsv/source_advance_search.php`. L'accès aux noms des listes permet d'obtenir les copies numérisées via un formulaire.

2.1.5 La question de la transcription

De part l'hétérogénéité des documents d'archives, du point de vue de la forme, du contenu, de la qualité originelle ou de l'état de préservation, l'éventualité d'une transcription inégale était envisageable.

Environ 75% des archives sont dactylographiées, 16% sont mixtes avec des informations manuscrites et 9% sont entièrement écrites à la main.

L'USHMM fournit un indicateur de lisibilité des documents.

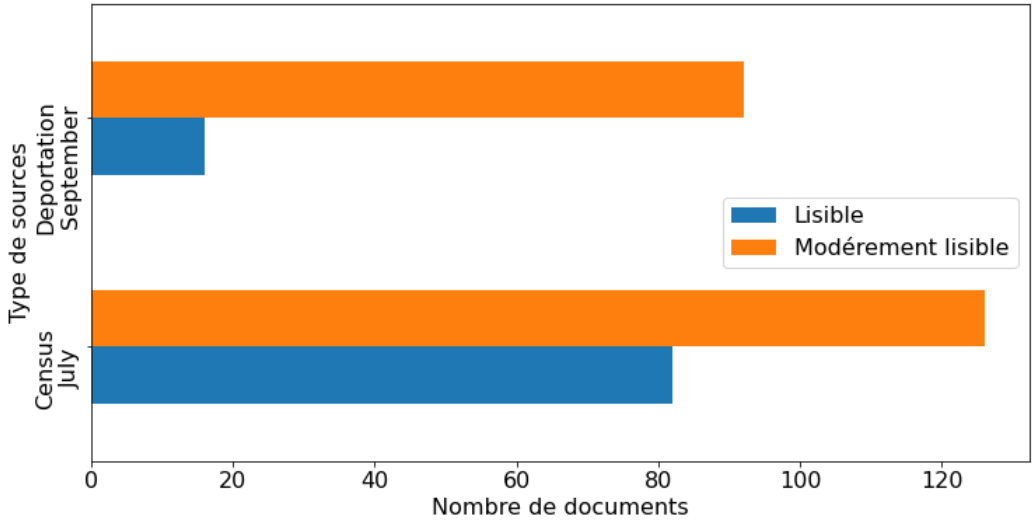


FIGURE 2.9 – Analyse de la lisibilité des sources (voir le code).

En dépit de ces difficultés, les données numérisées de l'USHMM sont assez remarquablement fidèles aux archives. Elles conservent fautes et accentuations erronées des originaux et parviennent à restituer les informations peu lisibles qui font sens.

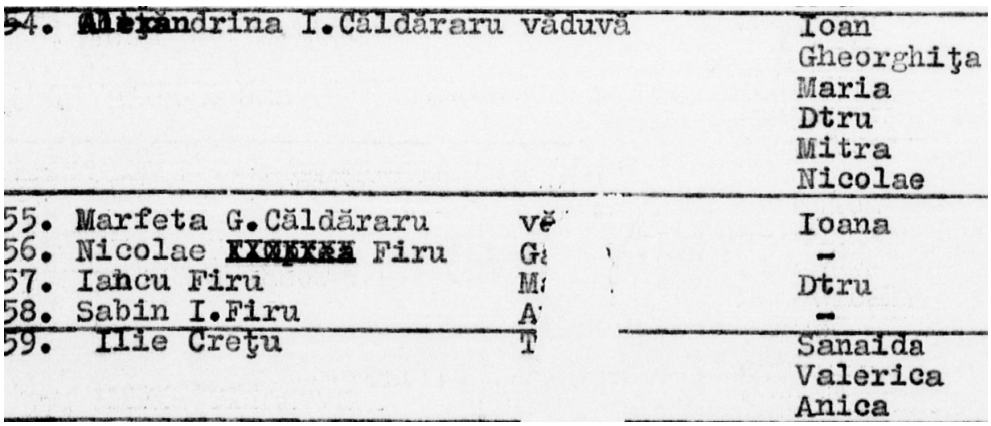


FIGURE 2.10 – Detail de liste[24].

24. RG-25.050M (File ID : 45869), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=45869` (visité le 10/09/2022), Copyright *Arhivele Naţionale ale României*.

Prénom	Nom	Statut Familial	Statut marital
Alexandrina [25] I	Caldăraru	Self	Widowed [26]
Ioan	Căldăraru	Child	
Gheorghița	Căldăraru	Child	
Maria	Căldăraru	Child	
Dtru	Căldăraru	Child	
Mitra	Căldăraru	Child	
Nicolae	Căldăraru	Child	
Marfeta G	Căldăraru	Self	Married
Vă ?? [27]	Căldăraru	Wife	Married
Ioana	Căldăraru	Child	
Nicolae [28]	Firu	Self	Married
G [29]	Firu	Wife	Married
Iancu	Firu	Self	Married
Ma ??	Firu	Wife	Married
Dtru	Firu	Child	
Sabin I	Firu	Self	Married
A ??	Firu	Wife	
Ilie	Crețu	Self	Married
T ??	Crețu	Wife	Married
Sanaida	Crețu	Child	
Valerica	Crețu	Child	
Anica	Crețu	Child	

TABLE 2.7 – Le tableau numérique correspondant à la figure 2.10

Les rares exceptions concernent en particulier les inscriptions qui chevauchent les lignes des tableaux, cette superposition ayant parfois pour effet de manquer un ou plusieurs noms de la case, ou un changement d'information (voir la note 18 du tableau 2.2).

Le deuxième cas de figure observé concerne certains renseignements dactylographiés et biffés ou commentés de manière manuscrite. Dans l'exemple ci-dessous, ces personnes ont tout de même été incluses (deux fois puisqu'elles l'étaient également aux lignes 131 et 132 du document).

FIGURE 2.11 – Detail de liste [30].

25. Le prénom est restitué malgré les ratures.
26. Veuve (*văduvă*).
27. Les lacunes sont indiquées par les points d'interrogation.
28. Les biffures ne sont pas prises en compte.
29. Cette entrée devrait être « G ?? », mais cela relève du détail.
30. RG-25.050M (File ID : 45835), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : `https://www.ushmm.org/online/hsv/source_view.php?SourceId=45835` (visité le 10/09/2022), Copyright *Arhivele Naţionale ale României*.

Cette phase d'analyse permet de dégager les spécificités de notre corpus et d'entrevoir les premières actions de standardisation, de correction et d'enrichissement à mener. En ce sens, cette étape procure également des clés pour le choix des solutions qui devront être mises en œuvre pour réaliser les traitements.

2.2 Préparation des données

2.2.1 Le choix des outils

2.2.1.1 Les outils disponibles

Malgré l'abondance de littérature sur le sujet, il apparaît que peu d'outils complets soient disponibles en accès libre (*open access*).

D'une part, les programmes mis au point pour les projets répondent largement aux besoins ponctuels et précis pour lesquels ils ont été conçus. Nous avons vu précédemment que les approches sont effectivement très dépendantes du type de données. Les auteurs détaillent parfois quelques-uns des algorithmes produits mais le plus souvent, le détail des opérations de traitement ne sont pas disponibles.

D'autre part, un certain nombre de solutions sont développées par le secteur privé, qui peut également intervenir en appui des projets de recherche.

Deux ressources proposent des listes de logiciels ou de modules en différents langages de programmation (principalement Python, R ou Java)[31]. Parmi eux, seulement cinq disposent d'une interface graphique (*graphical user interface* ou GUI).

Les solutions *OpenRefine* et *Dataiku* peuvent également être exploitées pour réaliser des couplage de données.

Concernant les calculs de distance, une multiplicité de modules existent, parmi lesquels de très diverses implémentations des algorithmes formels des différentes distances.

2.2.1.2 Les besoins du projet et premier arbitrage

L'objectif du travail relatif à l'appariement est double :

1. d'une part, de créer des outils qui permettent de qualifier l'appariement déjà réalisé, le compléter et le corriger, voire de l'automatiser en partie[32] ;
2. d'autre part, que ces solutions soient réalisées de telle sorte que leurs composants puissent être réutilisés et adaptés à de futurs travaux.

31. *Data-Matching-Software : A List of Free Data Matching and Record Linkage Software.* URL : `https://github.com/J535D165/data-matching-software` (visité le 06/09/2022) ; ropeladder, *Record Linkage Resources*, 24 août 2022, URL : `https://github.com/ropeladder/record-linkage-resources` (visité le 06/09/2022).

32. En particulier pour les futurs appariements.

Plus qu'un flux (semi-)automatisé d'appariement sur le corpus étudié lors de la phase courante du projet, il convient de mettre en place des modules qui puissent être recyclés, adaptés pour de futurs appariements, pour les phases ultérieures du projet ou sur d'autres projets.

Ainsi la standardisation et le nettoyage des données (suppression des diacritiques, des ponctuations, passage en casse uniforme) constituent des étapes qui peuvent largement être réutilisées.

La correction, l'enrichissement des variables, de même que l'indexation, les règles de blocage ou de classification, dépendront en revanche bien davantage de la disponibilité des données dans les autres sources qui viendraient à être étudiées dans le futur.

Du point de vue du contexte du projet, les solutions développées doivent être intelligibles, aisément modifiables et en adéquation avec les compétences de l'équipe projet.

En outre, la prise en main doit être relativement rapide afin de pouvoir fournir les différents livrables au prestataire en fonction des échéances planifiées :

— 20 avril 2022 : livraison d'un premier fichier d'individus appariés regroupés par famille ;
— mai 2022 : évaluation de la congruence des appariements ;
— juin 2022 : modélisation des données ;
— juillet 2022 : détermination des itinéraires de trains et géolocalisation, correction des datasets ;
— 25 juillet 2022 : initialisation de la phase d'importation des données dans Geovistory [33] ;
— fin août 2022 : chargement des données dans Geovistory [34], création des comptes.

Compte-tenu de ces contraintes, l'équipe effectue un premier arbitrage concernant le cadre de travail du projet.

2.2.1.3 Choix de l'environnement technique : le langage Python

En lien avec les ingénieurs de la plate-forme géomatique de l'EHESS, mais également eu égard aux programmes et langages utilisés par l'équipe mais également par le prestataire, l'arbitrage s'est porté sur un processus en Python [35], qui dispose de l'ensemble des modules nécessaires au travail envisagé et garantira la pérennité et l'adaptabilité des solutions développées dans le cadre de la mission.

Du point de vue de l'organisation, la plateforme géomatique de l'EHESS utilise

33. Voir la note 7
34. Voir la note 7
35. Langage de programmation interprété, voir `https://www.python.org/`

« Colaboratory », ou « Colab » [36] pour collaborer sur des projets de programmation et le suivi de stage.

Enfin, les données sont au format CSV. Les échanges avec le prestataire s'effectuent également dans ce format.

Aussi, le traitement des fichiers sera réalisé avec :

— Pandas [37], une bibliothèque écrite pour le langage de programmation Python, en ce qui concerne la manipulation et l'analyse des données ;
— après évaluation, un choix parmi les bibliothèques, programmes ou modules dédiés au calcul des distances et aux appariements.

Les spécifications de la bibliothèque Pandas sont disponibles à `https://pandas.pydata.org/docs/`.

Afin de pouvoir évaluer les solutions Python nécessaires à l'étude, la préparation des données constitue un préalable.

2.2.2 La standardisation des données

Il ressort de l'analyse des données que les actions suivantes seront nécessaires, que ce soit pour tester les outils, qualifier l'appariement ou évaluer son automatisation.

Notons que ces étapes devraient être exécutées dans cet ordre :

1. assurer la correspondance des encodages et des formats des fichiers et de variables à comparer, afin de pouvoir traiter les chaînes de caractères dans le format adéquat et ainsi éviter des erreurs dans le fonctionnement des procédures ;
2. nettoyer les chaînes de caractères, à savoir :
   (a) d'abord, supprimer les diacritiques pour non seulement optimiser les comparaisons futures, entre « Căldăras » et « Căldăraș » par exemple, mais aussi pour réduire l'éventail de signes à traiter dans les expressions régulières ultérieures ;
   (b) convertir en minuscule, pour les mêmes raisons ;
3. supprimer les ponctuations qui peuvent l'être à ce stade (voir ci-dessous) ;
4. corriger les entrées si possible (abréviations usuelles de prénoms) ;
5. segmenter les champs le cas échéant ;
6. enrichir les données le cas échéant (en particulier les genres).

36. « Colab » est un produit de Google Research qui permet d'écrire et d'exécuter un code Python dans un navigateur. Voir `https://research.google.com/colaboratory/faq.html?hl=fr`
37. Voir `https://pandas.pydata.org/`

2.2.2.1 Encodage et formats

Cette opération est réalisée lors de la lecture et de la création des dataframe Pandas à partir des datasets (voir le code B.1.1).

Les fichiers du projet ont une structure bien définie (voir 3.2.1.1), le fichier de l'USHMM étant une source originale immuable et les fichiers en provenance du prestataire correspondant à un schéma défini, on s'assure de les charger selon les mêmes conventions, à savoir :

1. au format UTF-8 (notamment du fait de l'accentuation de la langue) ;
2. au format CSV délimité par « ; », par convention et pour éviter de possibles erreurs avec les virgules présentes dans de multiples champs [38] ;
3. aucune colonne des fichiers n'est utilisée comme index du dataframe lors des chargements, une telle opération n'est effectuée que de manière réfléchie et volontaire à certains points du script ;
4. toutes les données sont chargées au format texte (à défaut, Pandas tente de déterminer le type de données seul et le résultat peut varier sur un même type de données, en fonction des fichiers) ;
5. une exception notable concerne les fichiers originaux de l'USHMM pour lesquels les âges peuvent contenir du texte comme évoqué plus haut (voir 2.1.4.1). Un convertisseur transforme ces entrées problématiques en chiffre (la fonction *Convert_Age(x)*).

En résumé, l'approche d'importation est assez stricte. L'expérience a montré que la bibliothèque Pandas ne parvient pas à fournir des résultats appropriés et constants lorsqu'elle est laissée autonome pour ces opérations.

Enfin, après avoir chargé les fichiers, une conversion des labels d'attributs est effectuée au moyen d'une table de conversion, afin d'assurer la concordance des variables [39] (voir la fonction *Change_Column_Names(df, dict)* en annexe B.1.2).

2.2.2.2 Standardisation des chaînes

L'objectif, entre autres, est de neutraliser les différences de casse, les variations de diacritiques sans perte d'information (voir le code B.1.3), au moyen du module *unidecodedata* [40] et d'expressions régulières dans la fonction *Normalise(df)*.

38. Certains fichiers originaux étaient corrompus de cette façon.
39. La table *standard_columns.xlsx*.
40. Voir `https://docs.python.org/fr/3/library/unicodedata.html`.

Les expressions régulières

Plutôt que de tenter de sélectionner les caractères à supprimer, les expressions régulières sont conçues pour ne sélectionner que les registres souhaités, à savoir l'alphabet latin non accentué obtenu après la normalisation effectuée à l'étape précédente.

Sont aussi conservées les espaces naturellement, et quelques ponctuations.

La question des signes de ponctuations

Les caractères « - », « ? » et « . » ont été maintenus. Le tiret d'abord car il intervient dans de nombreuses abréviations que le point suivant entend développer, une étape qui ne peut encore intervenir compte-tenu de l'état des chaînes à ce stade. Avec le point d'interrogation aussi, puisqu'ils remplacent des lettres illisibles dans les originaux. Ce point n'est pas entièrement neutre car ils sont comptabilisés comme un caractère dans les mesures de distances et, selon les mesures, les scores ne sont pas équivalents [41]. Néanmoins, nous souhaitions conserver cette information à la lecture des résultats.

Concernant le caractère point « . », il ne peut être supprimé sans pré-traitement dans les chaînes du type « N.C-tin ». Une espace doit être insérée afin de pouvoir segmenter « N » et « C-tin ». L'approche doit être différente des situations suivantes : « N. C-tin Dimitru » (espace déjà présente), « ..rtina » ou « Gheorgh... » (en lieu et place de caractères illisibles [42]). Ce point est donc traité ultérieurement par la fonction *Dot(df, var)*.

2.2.2.3 Développement des abréviations usuelles

Un certain de noms et prénoms abrégés peuvent faire l'objet d'un développement d'abréviations, comme « C-tin » en « Constantin ».

En revanche ce n'est pas toujours possible, lorsque de multiples orthographes et variantes par genre existent. Ainsi « Gh » peut correspondre à « Ghita », « Gheorgheta »,« Gheorghita », « Gheorghina ».

La conversion est effectuée par la table *spelling.xlsx* chargée et convertie en dictionnaire. Cette opération ne fait pas l'objet d'une fonction de part sa simplicité (voir le code B.1.5).

2.2.2.4 Segmentation des prénoms et noms

Afin de pouvoir analyser les éléments au niveau atomique, il est nécessaire de segmenter les informations pour les prénoms et noms qui comportent plusieurs mots ou groupes de mots.

41. Neutre pour Levenshtein, -5% à -20% pour les autres distances, en fonction du signe et de sa position.

42. Cas absent du recensement et de la déportation mais présent sur d'autres fichiers.

Il s'agit en particulier des surnoms, et des agrégations de noms, courantes dans le contexte historique et culturel (voir 2.1.4.1).

À l'aide d'expressions régulières, les surnoms introduits par *zis* ou *zisa* (« dit » et « dite » respectivement) sont extraits et retirés des prénoms ou noms par la fonction *Appellation(df, cols)* (voir le code B.1.4).

Les prénoms et noms multiples sont segmentés par la fonction *Split_Names(df, cols)*.

2.2.2.5 Enrichissements

Après avoir standardisé les données, il est possible d'apporter plusieurs enrichissements afin de faciliter les comparaisons.

Noms complets

Disposer du nom complet pourrait être intéressant pour rechercher et valider les paires lorsque ces informations sont identiques entre fichiers. C'est le rôle de la fonction *Fullname(df)* (voir le code B.1.6.1.

Années de naissance

L'analyse avait également révélé la disponibilité d'une quantité raisonnable d'âge des individus (variable dans le temps) mais de peu d'années de naissance.

En fonction de l'année de production du document d'archive fournie par l'utilisateur, la conversion des âges en années de naissance est réalisée par la fonction *Compute_Birthyear(df, doc_year : int, file)* (voir le code B.1.6.2).

Ainsi, un âge de 42 ans sur un document de 1942 permet de renseigner une année de naissance de 1900.

Le genre

Nous avions observé que le genre ne pouvait pas être déduit d'autres champs comme le statut familial ou marital (voir 2.1.4.1).

Néanmoins, des genres ont pu être inférés à partir des prénoms. La prévalence de ces derniers par genre a été analysée à partir de leur distribution dans un extrait normalisé de ces données, sur l'ensemble du corpus (plus de 130 000 entrées).

Une table *gender.csv* est ainsi obtenue et exploitée par la fonction *Derive_Gender(df)* (voir le code B.1.6.3).

Le gain est substantiel puisque cette opération permet de doubler le taux de couverture (voir le graphique ci-dessous).

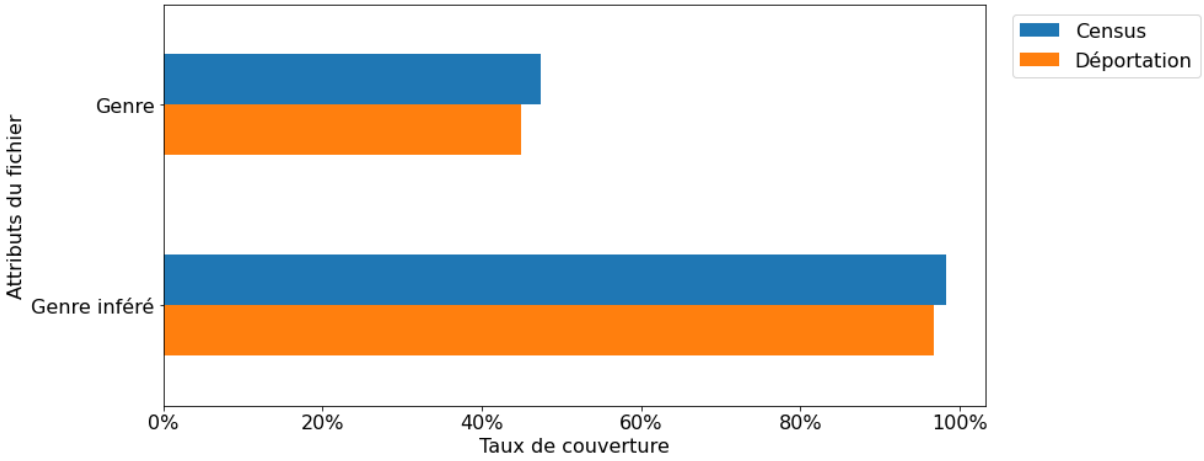


FIGURE 2.12 – Résultats de l'inférence des genres (voir le code).

En ce qui concerne la qualité de la donnée ainsi obtenue, la comparaison avec les données originales est positive et révèle un taux de correspondance important avec les informations des archives.

Par exemple, sur les 7 617 personnes de sexe féminin enregistrées dans les archives, le genre inféré est conforme pour 7 419 d'entre elles.

Census		
Genre	Genre inféré	Total
female	female	7419
	undetermined	119
	male	79
male	male	2510
	female	77
	undetermined	23
undetermined	male	7224
	female	2632
	undetermined	1466

TABLE 2.8 – Correspondance Genre et Genre inféré.

La distribution homme - femme est en revanche inverse entre les deux données genre (majorité de femmes dans les sources mais davantage d'hommes inférés).

Le sous-ensemble de 7 224 nouveaux « male » inférés est donc contrôlé. Les prénoms correspondent bien dans leur grande majorité à des prénoms indéniablement masculins.

La préparation des données achevée, les tests d'appariements peuvent être initiés. Dans le cadre du projet, avant de passer à l'indexation et aux phases ultérieures, deux étapes intermédiaires étaient requises : évaluer les outils de comparaison, puis les mettre à profit pour réaliser une première évaluation de la qualité du couplage manuel, afin que le prestataire puisse construire le prototype de la base de données.

2.3 Analyse des outils de calcul de distance

2.3.1 Les mesures de similarité

Pour rappel, les publications suggèrent divers algorithmes pour évaluer le degré de correspondance de chaînes de caractères (voir la section 1.2.2.3). Elles disposent toutes d'avantages et d'inconvénients et s'adaptent diversement aux corpus sur lesquels elles sont mises en œuvre, en fonction des données.

Aussi, nous avons dû réaliser des tests afin de comprendre quel(s) type(s) d'algorithme(s) utiliser sur nos recensements nominatifs.

Comme de nombreuses implémentations ont été produites, notre choix s'est arrêté sur la librairie *textdistance*[43] car elle propose d'une part son propre ensemble de calculs de distance et elle prend également en charge les librairies les plus utilisées dans le domaine du *fuzzy matching*.

Différents calculs de distance ont été réalisés sur les données appariées manuellement, avec pour double objectif :

— analyser le comportement des différents algorithmes sur les données du projet et évaluer lesquels seraient les plus appropriés pour notre étude ;
— obtenir une première vue d'ensemble de l'appariement manuel.

Conformément aux attentes, les différents scores témoignent de particularités différentes. Après analyse, une combinaison de mesures nous apparaît être la plus appropriée au regard de notre premier objectif (qualification de l'appariement manuel).

En effet, avec un Monge-Elkan sur le nom complet supérieur ou égal à 0.8, nous qualifions 3 728 des 4 391 paires, soit 86%.

Parmi ces paires :

— 1 993 sont des *matchs* exacts (Levenhstein) sur à la fois le prénom et le nom ;
— 370 *matchs* exacts sur les prénoms et moins de un caractère d'écart sur le nom ;
— 310 *matchs* partiels sur les prénoms et plus de un caractère différent sur le nom, validés après examen ;
— 108 correspondances parfaites après sélection du meilleur prénom parmi les différents prénoms enregistrés ;
— 134 inversions noms - prénoms ;
— 322 pour lesquels moins de un caractère d'écart existe sur le nom, validés après examen ;
— 290 validés avec un score de Jaro-Winkler supérieur à 80% ;

ce qui résulte en 201 paires, dont l'orthographe est très affectée mais les personnes correspondent pour moitié, le delta regroupant des cas discutables.

43. Voir : `https://github.com/life4/textdistance` et `https://pypi.org/project/textdistance/`

Parmi les 663 paires appariées dont le score Monge-Elkan reste inférieur à 0.8, 395 obtiennent un score Levenshtein de 0.8 et plus sur la meilleure association de prénoms. Il s'agit pour la plupart de paires valides, pour lesquels on observe deux cas de figure principaux.

D'une part, l'algorithme Monge-Elkan est négativement impacté par la présence d'une initiale supplémentaire en fin de nom complet. D'autre part, pour nombre de ces familles, un nom alternatif a été enregistré (les prénoms correspondent sur l'ensemble et seul le nom varie). Cela pourrait correspondre notamment à la sélection du nom du père dans un recensement et celui d'enfance de la mère pour la seconde liste.

Les 257 dernières personnes rapprochées sont vraisemblablement des erreurs pour la plupart, car peu de données correspondent, y compris les dates de naissance. Dans ce groupe, on observe également des individus qui, s'ils sont appariés correctement au niveau de la famille, les personnes ont été interverties entre elles.

En conclusion, le nombre d'appariements corrects d'environ 86% correspond aux attentes de l'équipe et satisfaisant, au regard des taux généralement obtenus sur ce type de données.

Sur cette base, des indicateurs sont produits sur les paires, pour l'équipe comme pour le prestataire :

— un score de Levenshtein sur le nom complet ;
— un score de Monge-Elkan sur le nom complet ;
— un dénombrement des individus de même nom dans la zone géographique pour chacun des fichiers (comptage des homonymes potentiels) ;

afin de disposer à l'avenir d'un indice de confiance sur les paires, comme pour réaliser les examens et corrections manuelles qu'il conviendrait d'apporter à l'appariement à une phase ultérieure.

2.4 Essai d'un processus de couplage

Pour évaluer si d'autres individus auraient dû être liés en revanche, un nouvel appariement est nécessaire [44]. Pour cette opération, initiée en fin de stage, le choix de la solution s'est porté sur la bibliothèque *Record Linkage* [45], dédiée au couplage et à la déduplication d'enregistrements dans ou entre des sources de données, pour lesquels elle fournit la plupart des outils nécessaires.

44. Voir le code B.2.2

45. J De Bruin, *Python Record Linkage Toolkit : A Toolkit for Record Linkage and Duplicate Detection in Python*, version v0.14, Zenodo, 1er déc. 2019, DOI : 10.5281/ZENODO.3559043, Voir également : https://github.com/J535D165/recordlinkage et https://pypi.org/project/recordlinkage/.

2.4.1 Indexation et blocage

Record Linkage permet d'indexer les sources, dans leur totalité, ou en pré-définissant des clés de blocage (voir la section 1.2.2.2).

Concernant le recensement de juillet et de la déportation, les fichiers comportent respectivement 21 549 et 13 033 enregistrements, soit 280 848 117 comparaisons à réaliser sans clé de blocage.

Par conséquent, la définition de telles clés s'impose. Sur ce sujet, un très grand nombre de possibilités existent puisqu'il est possible d'utiliser un attribut seul pour servir de clé, mais également toute combinaison d'attributs ou subdivision d'attributs. Sans faire une liste exhaustive, les clés de blocage usuelles sont :

1. le genre (les individus de même sexe seulement sont comparés) ;
2. les éléments d'adresses, en particulier ceux qui sont standardisés comme le code postal, des noms de villes normalisés ;
3. des années de naissance ;
4. de manière générale, tout élément qui permet de distinguer des groupes plus restreints à comparer sur la base d'éléments invariants ;
5. des unions de tout ou partie de ces éléments.

Dans le contexte du projet, une clé de blocage en particulier nous a intéressé. Il s'agit du code des localités mis en place pour le projet. Comme nous l'avons rappelé, les deux recensements sont proches et peu de changements, dans les familles ou adresses, sont attendus entre ces deux événements.

L'ajout de cette clé de blocage permet de restreindre le nombre de comparaisons à 4 861 124 : seuls les individus enregistrés dans la même localité seront comparés les uns aux autres.

Une clé de blocage sur à la fois le code localité et le genre réduit l'index des paires à 2 528 167.

Sur cette dernière base, nous définissons les règles de comparaison des enregistrements.

2.4.2 Comparaison des paires et classification

Pour rappel (voir la section 1.2.2.4), cette étape permet de choisir les attributs (valeurs) qui seront rapprochés et testés selon les critères choisis, par exemple une similarité supérieure à 0.7 (soit de 70%).

Les premiers tests effectués avec un nombre limité de règles ont montré qu'il est alors relativement difficile de catégoriser les paires liées de celles non liées.

En effet, avec une comparaison sur les seuls prénoms, noms et date de naissance par exemple, pour toutes les paires sans cette dernière information, le résultat consiste en une liste de *matchs* potentiels dont les noms et prénoms ont tous atteint les seuils de similarité. Il est alors peu aisé de distinguer les paires les plus probantes :

Index		Fichier 1		Fichier 2	
ID Fichier 1	ID Fichier 2	Nom	Année naissance	Nom	Année naissance
8750260	8799761	bozsodi susana	1934	dondos susana	
8750282	8799761	dondos elisabeta	1929	dondos susana	
8750283	8799761	dondos ghizela	1930	dondos susana	
8750282	8799762	dondos elisabeta	1929	dondos iuliu	
8750283	8799762	dondos ghizela	1930	dondos iuliu	
8750291	8799762	dondoczi semoil	1923	dondos iuliu	
8750361	8799762	dondoczi vasile	1924	dondos iuliu	
8750251	8799789	dondos iuliu	1934	dondos rozalia	
8750291	8799789	dondoczi semoil	1923	dondos rozalia	
8750361	8799789	dondoczi vasile	1924	dondos rozalia	
8750251	8799790	dondos iuliu	1934	dondos paraschiva	
8750361	8799790	dondoczi vasile	1924	dondos paraschiva	
8750251	8799792	dondos iuliu	1934	dondos elisabeta	
8750361	8799792	dondoczi vasile	1924	dondos elisabeta	
8750251	8799793	dondos iuliu	1934	dondos ghizala	
8750291	8799793	dondoczi semoil	1923	dondos ghizala	
8750361	8799793	dondoczi vasile	1924	dondos ghizala	
8750251	8799794	dondos iuliu	1934	dondos rozalia	
8750282	8799794	dondos elisabeta	1929	dondos rozalia	
8750283	8799794	dondos ghizela	1930	dondos rozalia	
8750291	8799794	dondoczi semoil	1923	dondos rozalia	
8750361	8799794	dondoczi vasile	1924	dondos rozalia	
8750251	8799796	dondos iuliu	1934	dondos ileana	
8750283	8799796	dondos ghizela	1930	dondos ileana	
8750291	8799796	dondoczi semoil	1923	dondos ileana	
8750315	8799796	fodor ileana	1936	dondos ileana	
8750361	8799796	dondoczi vasile	1924	dondos ileana	

TABLE 2.9 – Exemple simplifié de résultats de comparaison.

Un seul individu du fichier 2 (surligné en rouge) est rapproché avec plusieurs individus du fichier 1, autant de fois que les similarités définies sont respectées. Sans plus de critères, il est peu commode d'identifier les meilleures paires.

Notons que le résultat ci-dessus est obtenu sur la base de seuils de similarité Jaro-Winkler de 0.7 (prénom) et 0.8 (nom), ce qui semble trop flou et permissif.

Afin d'obtenir davantage de critères discriminants, les règles suivantes peuvent être définies comme ci-dessous (parmi les premiers tests effectués) :

— prénom 1 : similarité de Levenshtein >= 0.7 ;
— prénom 2 : similarité de Levenshtein >= 0.7 ;
— nom : similarité de Levenshtein >= 0.8 ;
— nom complet : similarité de Levenshtein >= 0.8 ;
— année de naissance : comparaison exacte [46] ;
— identifiant famille : comparaison exacte [47] ;
— genre inféré : comparaison exacte [48] ;
— statut familial : comparaison exacte ;
— identifiant localité : comparaison exacte [49].

Les paramètres ci-dessus procurent les performances suivantes :

Mesure	Résultat
Paires indéxées	2 092 398
Précision	0,056
Rappel	0,809
F1-Score	0.105
Durée	5 minutes

TABLE 2.10 – Résultats d'un test déterministe (règles à base de seuils).

Le couplage produit 6 227 paires potentielles (dont des propositions multiples pour un individu comme illustré dans la table 2.9 ci-dessus).

Parmi elles, 3 426 sont également liées dans l'appariement manuel et 2 524 offrent le même couplage. Notons que la vaste majorité (2 466) de ces paires communes recoupent les 86% de paires jugées satisfaisantes dans l'analyse du couplage manuel.

Pour restreindre encore le nombre de paires comparées et limiter la recherche aux enregistrements voisins, il est notamment possible d'utiliser la méthode des plus proches voisins (*Sorted Neighbourhood*). Le test est réalisé en mode d'apprentissage automatique non supervisé.

L'indexation est paramétrée en *Sorted Neighbourhood* sur le nom de famille, avec des clés de blocage sur le code localité et le genre inféré.

Le nombre de paires candidates est drastiquement ramené à 53 988 couples à classer.

46. Il est en possible de tester les valeurs numériques avec une tolérance, mais cette option engendrait un dépassement des capacités de mémoire.
47. Ceci n'est possible que sur la réconciliation d'un corpus préalablement apparié, ce qui était le cas dans cet essai.
48. Le genre étant une clé de blocage, les valeurs correspondent, néanmoins on peut ici clarifier que des valeurs manquantes ne constituent pas un résultat positif.
49. Idem.

Les résultats de cet appariement ont été les suivants :

Mesure	Résultat
Paires indéxées	53 988
Précision	0,346
Rappel	0,960
F1-Score	0.509
Durée	1-2 minutes

TABLE 2.11 – Résultats d'un test *Sorted Neighbourhood*.

Cet appariement produit 5 692 paires potentielles (dont des propositions multiples pour un individu).

Parmi elles, 3 162 sont également liées dans l'appariement manuel et 2 348 offrent le même couplage, avec 2 295 de ces paires communes situées dans la tranche des 86% de paires jugées satisfaisantes dans l'analyse du couplage manuel.

Ces résultats sont intéressants dans la mesure où ils permettent de mesurer la performance de ce modèle sans entraînement préalable et donc d'évaluer son comportement sur des données nouvelles.

Sur ces paires, le nombre de correspondances entre les neuf attributs comparés est détaillé dans le tableau ci-dessous :

Correspondances	Appariement manuel	Modèle (paires potentielles)	Modèle (paire uniques [50])
9	21	23	16
8	425	533	423
7	1384	1789	1348
6	1011	1448	742
5	782	1669	266
4	495	230	38
1-2-3	273	-	-
Total	4391	5692	2833

TABLE 2.12 – Concordance des attributs par appariement.

Ces résultats donnent une idée du taux de confiance qui peut être accordé aux différentes paires. En effet, plus le nombre d'attributs correspondants est élevé, plus les deux enregistrements sont proches et donc susceptibles de référer à la même personne.

50. La mesure est donnée à titre indicatif uniquement et effectuée sur les enregistrements pour lesquels une seule paire est proposée : pour ceux proposés dans de multiples combinaisons, les résultats ne sont pas connus avant d'avoir arrêté un choix de paire, le cas échéant.

En résumé, il nous a été possible d'estimer la qualité de l'appariement réalisé manuellement et l'équipe l'a jugé satisfaisante à ce stade de la recherche. Le couplage peut encore être affiné, notamment par des corrections continues toujours en cours mais aussi par l'apport des informations additionnelles qui seront obtenues des rapprochements futurs avec les autres sources disponibles, en provenance des camps d'une part et d'autres recensements d'autre part.

Nous avons pu également tester des solutions de couplage programmées en Python avec *Record Linkage*, dont certains résultats semblent insuffisants, d'autres prometteurs. Cette partie du travail reflète bien la difficulté de l'exercice du couplage de données relayée par la littérature. Nous discutons des perspectives d'amélioration et présentons le résultat matériel et concret du projet dans le prochain chapitre.

Chapitre 3

Perspectives et réalisations concrètes

3.1 Perspectives d'amélioration

3.1.1 Fouille et segmentation

Nous avons vu que les sources sont parfois pauvres en informations et que la graphie des noms est extrêmement variable.

Une fouille et une segmentation plus poussées permettraient de réduire les informations parasites et faire mieux ressortir celles qui sont pertinentes, un travail entamé mais qui nécessite d'être poursuivi :

- extraire les prénoms des chefs de famille ;
- isoler et éliminer les initiales correspondantes imbriquées dans les prénoms ;
- identifier les initiales communes à plusieurs membres d'une famille afin de les traiter séparément, même en l'absence du prénom du père ;
- dresser les listes des patronymes (pour comparer les noms alternatifs possibles) et prénoms au niveau des familles ;
- convertir les âges en tranches d'âges pour autoriser une comparaison floue ;
- enrichir le statut familial en fonction de l'âge (par exemple attribuer la valeur « Child » aux personnes de moins de 16 ou 17 ans, là où l'information est actuellement manquante) ;
- poursuivre la normalisation des fautes orthographiques sur les prénoms ;
- évaluer la pertinence de subdiviser la table de correction orthographique par genre, afin d'introduire davantage de développements d'abréviations ;
- considérer le développement ou la normalisation des « i » en « ioan » des « g », « gh » et « ghe » en « gheorghe » ou « gheorghita », les deux formes les plus courantes ;
- compléter la table d'attribution des genres pour les prénoms pour lesquels l'analyse n'a pas identifié de résultats probants.

Cette liste de pistes non exhaustive reflète bien l'ampleur des possibilités, à mettre

en perspective avec les attentes du projet et les ressources disponibles pour effectuer ces développements.

3.1.2 Optimiser les modèles

3.1.2.1 Renforcer l'indexation

De la même façon, le modèle en lui même peut largement gagner en efficacité. Développé en fin de stage, il n'a pas pu bénéficier du nombre de tests et d'ajustements nécessaires à de meilleures performances.

Concernant l'indexation d'abord, les possibilités sont aussi diverses qu'en matière de fouille et de segmentation. La création de nouvelles clés de blocage plus ou moins strictes selon les informations à apparier est par exemple envisageable.

Les informations des listes des camps notamment sont extrêmement limitées. Pour celles-ci, l'introduction de clés de blocage sur une ou plusieurs des initiales pourrait réduire le nombre de faux positifs.

Pour faire un parallèle avec la question des attributs discutée plus haut, les nouvelles données extraites des enregistrements peuvent constituer autant de clés efficaces, à l'instar des tranches d'âge.

Bénéficier de davantage de *blocking keys* peut autoriser des seuils de comparaison plus modérés ou une augmentation de la fenêtre de recherche dans l'approche des voisins triés.

3.1.2.2 Diversifier les comparaisons

La problématique est la même pour la comparaison. Avec neuf attributs actuellement, les possibilités sont limitées. Introduire de nouvelles entrées discriminantes à analyser fournirait autant d'indices supplémentaires dans le choix des vrais positifs.

Deux autres leviers peuvent être mis en œuvre pour aider à la classification. En premier lieu, lors de l'agrégation des points de concordances (voir le tableau 2.12), une pondération différenciée peut être utilisée : une même année de naissance est plus significative que la correspondance d'une tranche d'âge, elle même plus révélatrice qu'une ressemblance d'un deuxième prénom.

Ensuite, une approche en plusieurs passages, progressivement plus permissifs, contribuerait à simplifier l'analyse des résultats en dégageant en priorité les paires fortement liées vers les cas moins éloquents.

3.1.2.3 Perfectionner les modèles

Enfin, à propos des modèle en eux-mêmes, la conception de règles additionnelles et sur mesure renforcerait leur efficacité.

À titre d'exemple, une comparaison croisée et systématique des prénoms et initiales pourrait être bénéfique.

En outre, la mesure de distance utilisée pour estimer les similarités de l'appariement manuel n'est pas disponible dans la bibliothèque *Record Linkage*. Le processus accepte cependant l'intégration de fonctions personnalisées, un biais par lequel la mesure de Monge-Elkan pourrait être implémentée.

D'autres modèles sont également à disposition dans la bibliothèque *Record Linkage* comme la régression logistique, les *SVM*, autant d'alternatives possibles qui reflètent bien l'une des problématiques discutées dans la littérature, à savoir la difficulté de trouver la méthode la plus adaptée parmi l'offre très diversifiée d'algorithmes.

3.1.2.4 Développement du code Python

Le code conçu au cours du projet comporte des limites à prendre en compte.

Premièrement, il a été produit pour répondre à un besoin particulier en fonction des diverses échéances du projet. En conséquence, il n'a pas été développé comme un outil générique. Aussi, il ne peut en l'état s'adapter à différents corpus ou fonctionner sur des données non conformes. Il est basé sur la structure des données de l'USHMM. Les labels de fichiers, d'attributs, le format de données sont intimement liés au schéma des sources CSV et des modèles mis en place avec le prestataire.

Ensuite, il n'est pas construit comme une chaîne de traitement à proprement parler. S'il assure les différentes transformations les unes après les autres, il ne s'agit pas d'une application qui permettrait de sélectionner des fichiers en entrée pour disposer d'un dépôt de résultats au format CSV (ou autres), d'une interface de consultation, d'indicateurs. Il s'agit d'un programme en code, d'aide à la décision. Une des améliorations les plus immédiates serait d'ajouter des variables à certaines fonctions, notamment celle exploitant *Record Linkage*, dans la mesure du possible, puisque la quantité d'options devrait être soigneusement choisie au regard du nombre important de possibilités de paramétrage.

L'ajout d'une interface est techniquement réalisable avec *Record Linkage* (Record-Linkage Annotator [1]). Cependant, comme la plupart des systèmes d'annotation ou de validation [2] ne présentent pas les enregistrements *dans leur contexte*, il est malaisé de classer des paires dans ces conditions. Sans afficher les données voisines permettant de lier ou non des paires, il faut souvent retourner à la source. Dans le contexte du projet, un examen des numérisations s'est révélé bien souvent nécessaire.

1. Voir : `https://github.com/J535D165/recordlinkage-annotator`
2. Notamment ceux utilisés pour les entraînements.

3.2 Les résultats concrets pour le projet

3.2.1 Structurer et modéliser les données

Dans le cadre d'un projet ambitieux comme *La déportation des Roms en Transnistrie, 1942 - 1944. Trajectoires individuelles et destin collectif*, qui doit permettre des études statistiques et l'analyse des statuts administratifs, des variables sociodémographiques et des parcours dans la dimension spatiale, la question de la définition des données, de leur modélisation est une étape indispensable.

3.2.1.1 Le modèle de données

Afin de pouvoir exploiter massivement les documents identifiés, un modèle de données a été conceptualisé par l'équipe de recherche.

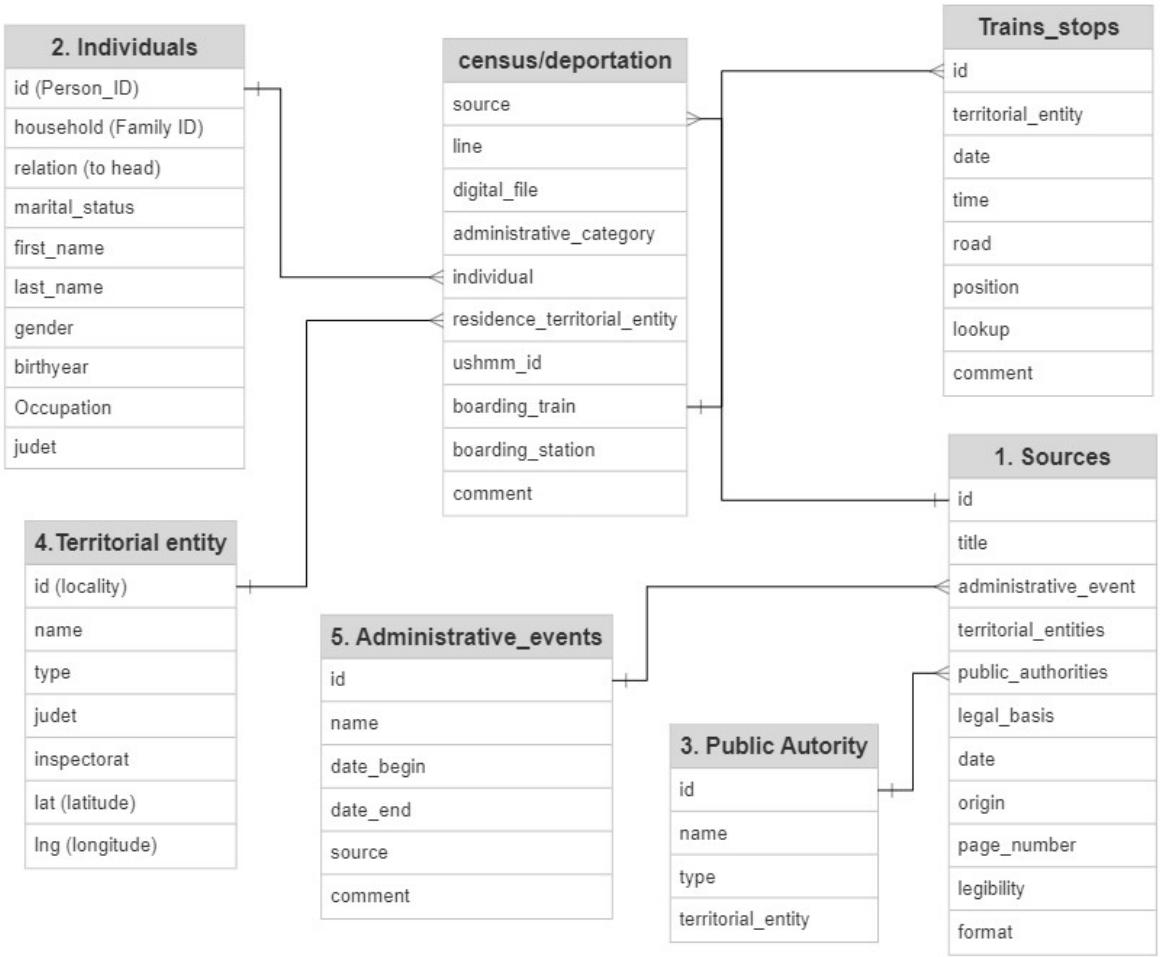


FIGURE 3.1 – Modèle entités - associations.

En reprenant les informations de la documentation afin qu'elle puisse être exploitée, l'architecture de ces données met en relation cinq entités principales dans le modèle ontologique suivant :

1. les documents nominatifs ;
2. les individus ;
3. les autorités publiques ;
4. les entités territoriales ;
5. les évènements administratifs.

Le projet a été l'occasion de clarifier les définitions, les types de données afin que le prestataire puisse modéliser, renseigner les ontologies et intégrer les données dans son environnement.

3.2.1.2 Premières analyses de faisabilité

Un point clé du projet étant de permettre de reconstituer les parcours individuels et collectifs, une géolocalisation et une analyse temporelle des trains de déportation peut être réalisée, grâce aux sources mentionnant les horaires de départ, de point de passage et d'arrivée dans les diverses gares.

L'organisation de la déportation est intimement liée à l'organisation territoriale de la Roumanie de 1942. C'est un point particulier que le projet entend étudier en détail.

Ainsi, grâce aux données collectées, géocodées, le prestataire a pu procéder à des premières analyses qui laissent entrevoir un potentiel important pour la suite :

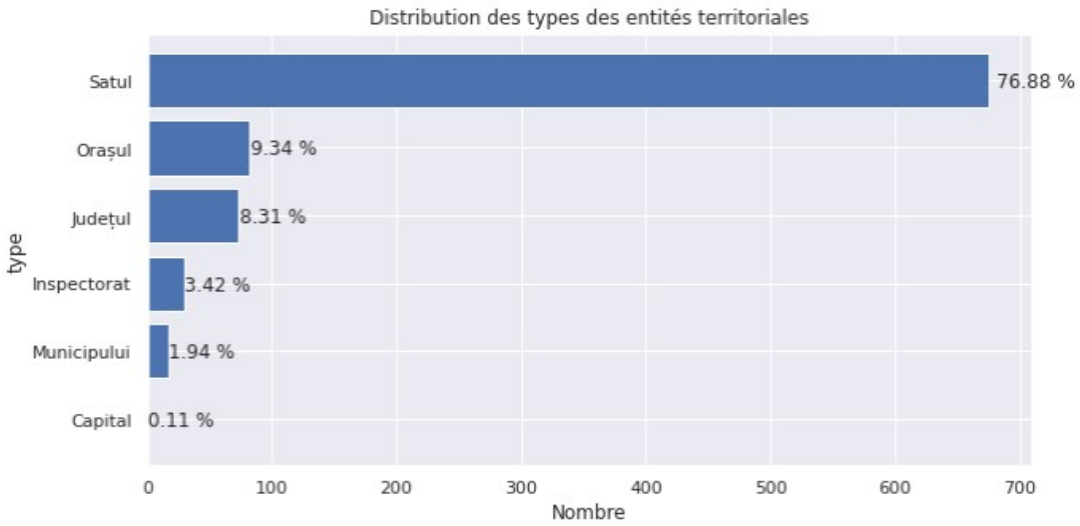


FIGURE 3.2 – Types d'entités territoriales[3].

3. Préparé par Gaétan Muck, KleioLab GmbH.

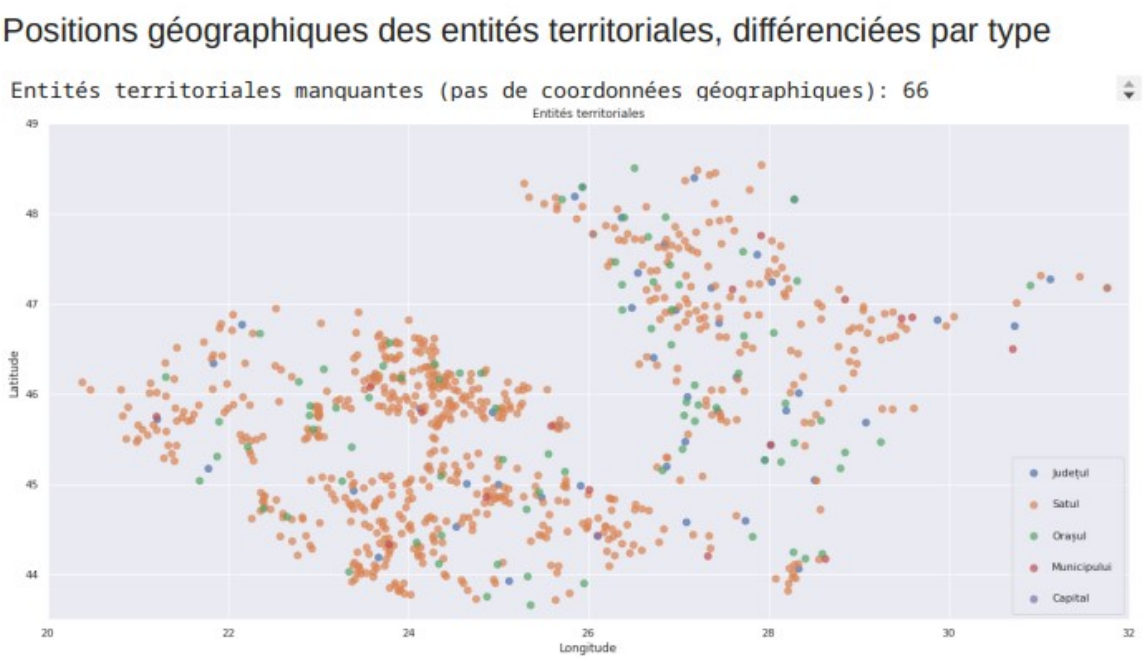


FIGURE 3.3 – Cartes des entités territoriales [4].

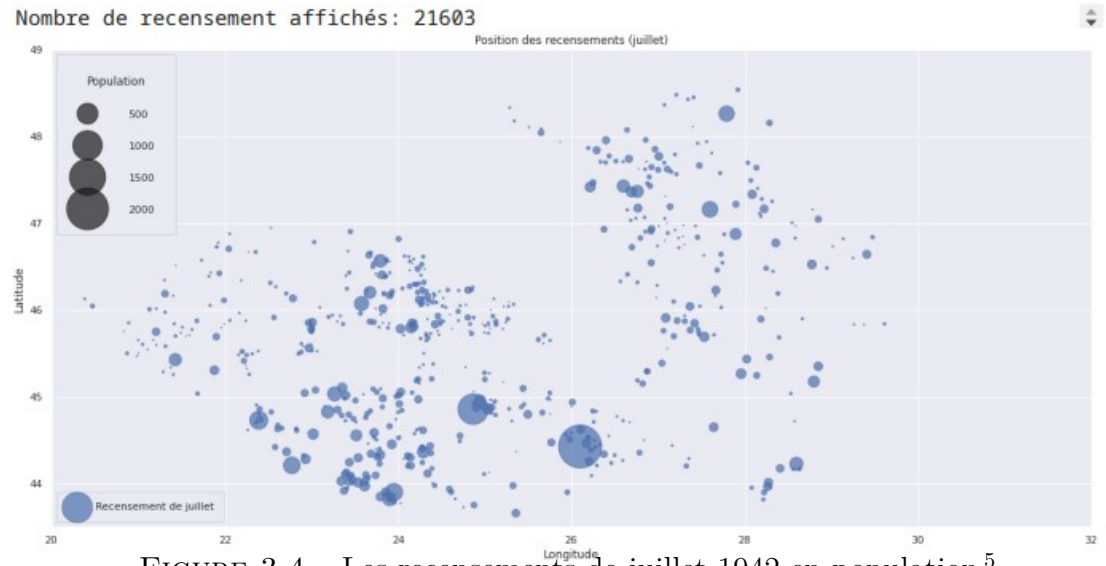


FIGURE 3.4 – Les recensements de juillet 1942 en population [5].

4. Préparé par Gaétan Muck, KleioLab GmbH.
5. Id.

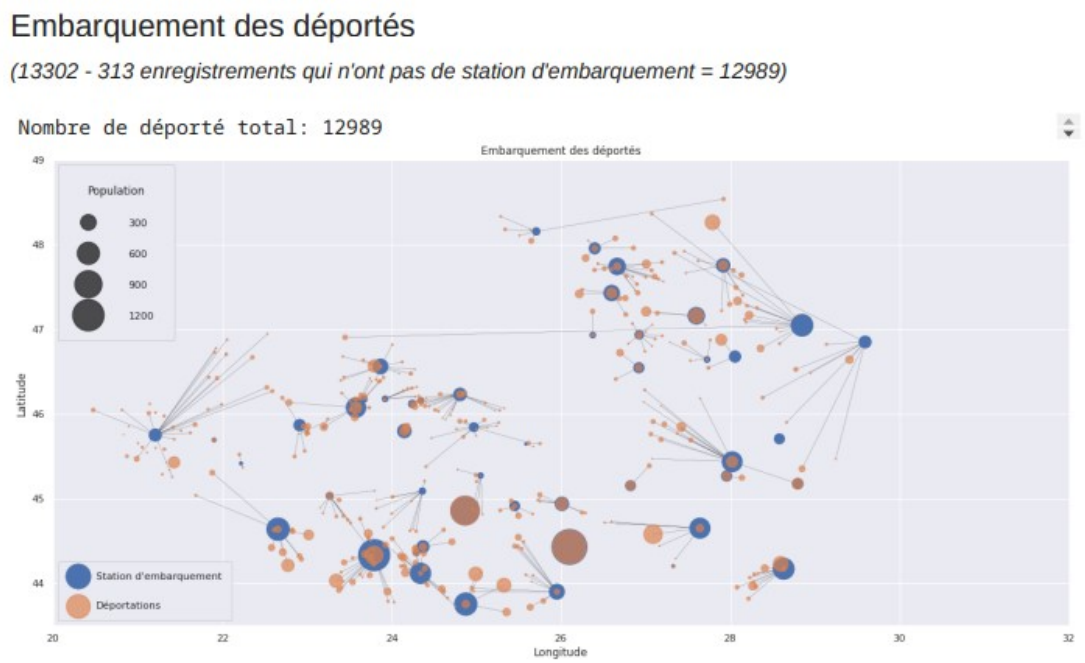


FIGURE 3.5 – Carte de la déportation [6].

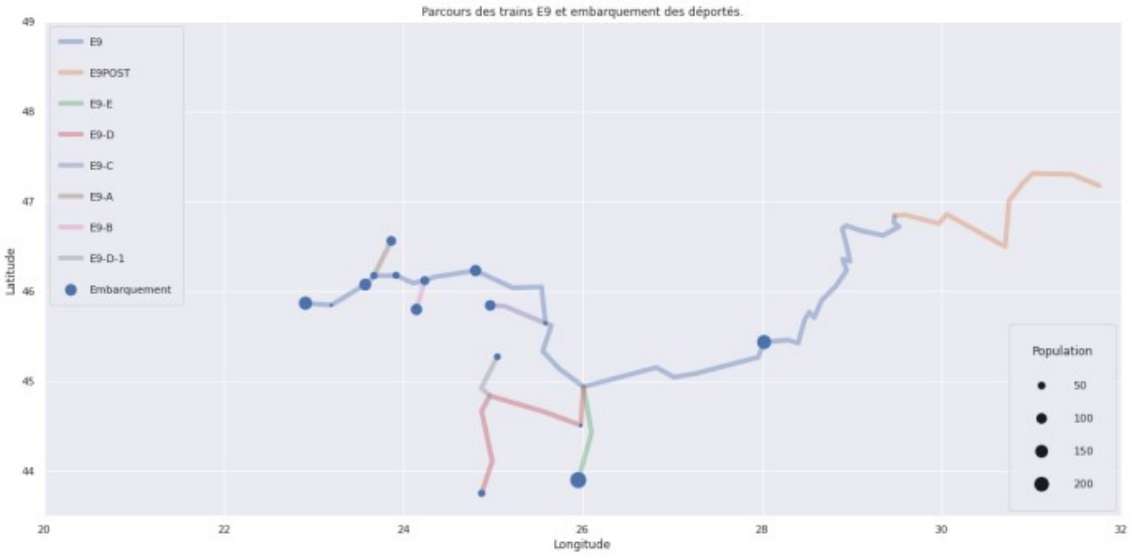


FIGURE 3.6 – Trajet d'un train et ses effectifs dynamiques [7].

En élargissant la chronologie, depuis les lieux d'origine jusqu'au retour des personnes, une visualisation des parcours permet une analyse à différentes échelles. Au niveau d'une personne ou des membres d'une famille, de personnes originaires du même lieu, l'étude pourra faire ressortir des faisceaux de parcours, des trajectoires typiques ou, au contraire, mettre en lumière des cas limites intéressants.

6. Préparé par Gaétan Muck, KleioLab GmbH.
7. Id.

Conclusion

L'étude de la déportation des Roms est un projet multiscalaire portant sur un vaste corpus (125 000 entrées individuelles pour environ 1 500 documents). L'historiographie, récente mais riche, de cet événement est surtout centrée sur l'appareil d'État. Orienté autour des parcours des individus à différentes échelles, le projet se donne pour ambition de suivre effectivement les personnes, de faire une histoire totale, à la fois fine et globale, communauté par communauté, localité par localité.

Un des points délicats d'une telle étude, qui détermine la qualité de l'ensemble de l'enquête est la duplication des identités entre les documents. Aussi, il faut établir une table de liens entre les personnes enregistrées dans les différentes archives administratives, en s'appuyant sur les noms, prénoms et en évaluant au regard des autres informations comme l'âge, les noms des membres de la famille ou encore le lieu de résidence s'il s'agit des mêmes personnes.

Cette entreprise d'appariement est rendue délicate par la nature de l'information elle-même. En effet, les données nominatives sont souvent très variables et le sont encore davantage sur les documents historiques. Les agents en charge de constituer les listes, les personnes elles-mêmes ne sont pas toujours constantes dans la manière de s'identifier ou de préciser un âge, surtout dans des territoires où l'état civil n'est pas encore fortement établi.

Aussi, l'étude s'est confrontée à des données relativement inégales, imprécises mais surtout dénaturées par la présence de nombreuses abréviations et initiales d'une part et par la récurrence de certains prénoms ou noms très communs d'autre part.

Néanmoins, nous avons pu évaluer la qualité des rapprochements effectués manuellement et ainsi permettre de poursuivre le développement du projet.

Par ailleurs, nous sommes parvenus avec un succès relatif à développer en fin de mission un outil d'appariement codé en langage Python qui semble prometteur, même s'il nécessiterait davantage de tests et d'ajustements.

Ce modèle a permis de confirmer la qualification des couplages manuels et des essais effectués sur les prochaines listes à apparier ont montré la capacité de l'outil à identifier, si ce n'est une personne précise, en tout cas des groupes d'individus proches, facilitant les recoupements manuels. Il devrait en résulter un gain de temps non négligeable pour l'équipe de recherche dans les phases ultérieures du projet.

Annexes

Annexe A

Codes d'analyse

A.1 Analyse de la structure de la source CSV

Voir le graphique.

```
# modules import
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

# files path definition
drive.mount('/content/drive', force_remount=True)
entree_xml = "/content/drive/MyDrive/"

# set dataframe display options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

# source csv import
ushmm = pd.read_csv (entree_xml+'/personUSHMM.csv', encoding='utf8',sep=';'
                                    , index_col=False, dtype = str)

# control imported data
# print(ushmm.info())

# get columns coverage in %
percent_coverage = ushmm.notnull().sum() * 100 / len(ushmm)

# set font size
```

```
plt.rcParams.update({'font.size': 16})

# moving legend
#plt.legend(loc='center right')

# naming the x-axis
plt.xlabel('Taux de couverture')

# naming the y-axis
plt.ylabel('Attributs du fichier')

# plot % coverage by column in horizontal bars
percent_coverage.plot.barh(figsize=(6,15))

# shortening labels (get text after '-' if any)
ax = plt.axes()
def wrap_labels(ax):
    labels = []
    for label in ax.get_yticklabels():
        text = label.get_text()
        labels.append(text.split('-')[1] if '-' in text else text)
    ax.set_yticklabels(labels, ma ='center')
wrap_labels(ax)

# desired format of the ticks, here '40%'
fmt = '%.0f%%'
# format the x axis
xticks = mtick.FormatStrFormatter(fmt)
ax.xaxis.set_major_formatter(xticks)

# reverse order y axis
ax.invert_yaxis()

# save the plot as image
plt.savefig(entree_xml+'/ushmm_content.png', bbox_inches='tight')

plt.show()
```

A.2 Analyse de la structure des fichiers census et déportation

Voir le graphique.

```
# modules import
```

```
import pandas as pd
from google.colab import drive
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

# files path definition
drive.mount('/content/drive', force_remount=True)
entree_xml = "/content/drive/MyDrive/"

# set dataframe display options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

# source csv import
ushmm = pd.read_csv (entree_xml+'/personUSHMM.csv', encoding='utf8',sep=';'
                                        , index_col=False, dtype = str)
census = pd.read_csv (entree_xml+'/census.csv', encoding='utf8',sep=';',
                                        usecols = ['ushmm_id'],  dtype = str)
deportation = pd.read_csv (entree_xml+'/deportation.csv', encoding='utf8',
                                        sep=';', usecols = ['ushmm_id'],
                                        dtype = str)

# define irrelevant columns
drop_columns = ['Restricted','PersonType','Honorific','MiddleName','Suffix'
                                        , 'Description','pePlaceBirth-Place
                                        of Birth','peNationality-Nationality'
                                        ,'peMediaPrimary-Primary Media']

# filter individuals concerned by respectively the census and the
                                        deportation files
ushmm_census = ushmm[ushmm['PersonId'].isin(census['ushmm_id'])]
ushmm_deportation = ushmm[ushmm['PersonId'].isin(deportation['ushmm_id'])]

# remove irrelevant columns
ushmm_census.drop(drop_columns, axis=1, inplace=True)
ushmm_deportation.drop(drop_columns, axis=1, inplace=True)

# get columns coverage in %
cs_percent_coverage = ushmm_census.notnull().sum() * 100 / len(ushmm_census
                                        )
dpt_percent_coverage = ushmm_deportation.notnull().sum() * 100 / len(
                                        ushmm_deportation)
```

```
# set % series as dataframes
census_percent_coverage = pd.DataFrame(cs_percent_coverage, columns=['
                                    Pourcentage']).reset_index()
deportation_percent_coverage = pd.DataFrame(dpt_percent_coverage, columns=[
                                    'Pourcentage']).reset_index()
# insert source type to prepare pivot table
census_percent_coverage.insert(1, 'Fichier', 'Census')
deportation_percent_coverage.insert(1, 'Fichier', 'Déportation')
# rename columns to prepare pivot table
census_percent_coverage.rename(columns = {'index': 'Attributs'}, inplace =
                                    True)
deportation_percent_coverage.rename(columns = {'index': 'Attributs'},
                                    inplace = True)

# merge the 2 dataframes
scope_coverage = pd.concat([census_percent_coverage,
                                    deportation_percent_coverage])
# pivot table to create the graph
scope_content = scope_coverage.pivot_table(index="Attributs", columns="
                                    Fichier", values="Pourcentage",
                                    aggfunc=np.sum, sort=False)

# set font size
plt.rcParams.update({'font.size': 16})

# plot % coverage by column in horizontal bars
scope_content.plot.barh(figsize=(6,15))

# moving legend
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left')

# naming the x-axis
plt.xlabel('Taux de couverture')

# naming the y-axis
plt.ylabel('Attributs du fichier')

# shortening labels (get text after '-' if any)
ax = plt.axes()
def wrap_labels(ax):
    labels = []
    for label in ax.get_yticklabels():
        text = label.get_text()
        labels.append(text.split('-')[1] if '-' in text else text)
    ax.set_yticklabels(labels, ma ='center')
wrap_labels(ax)
```

```
# desired format of the ticks, here '40%'
fmt = '%.0f%%'
# format the x axis
xticks = mtick.FormatStrFormatter(fmt)
ax.xaxis.set_major_formatter(xticks)

# reverse order y axis
ax.invert_yaxis()

# save the plot as image
plt.savefig(entree_xml+'/scope_content.png', bbox_inches='tight')

# function to show the plot (needs to be after saving)
plt.show()
```

A.3 Analyse de la lisibilité

Voir le graphique.

```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import textwrap

drive.mount('/content/drive', force_remount=True)
entree_xml = "/content/drive/MyDrive/"

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

sources = pd.read_csv (entree_xml+'/sources.csv', encoding='utf8',sep=';')
sources.rename(columns={'legibility': 'Lisibilité'}, inplace=True)
sources_legibility = sources.groupby('administrative_event')['Lisibilité'].
                                    value_counts().unstack()

sources_legibility.rename(columns={'Easily Legible Text': 'Lisible',
'Moderately Legible Text': 'Modérement lisible'}, inplace=True)

# control transformations
print(sources_legibility)

# set font size
```

```
plt.rcParams.update({'font.size': 16})

# Create horizontal bars
sources_legibility.plot.barh(figsize=(12,6))

# Moving legend
plt.legend(loc='center right')

# naming the x-axis
plt.xlabel('Nombre de documents')

# naming the y-axis
plt.ylabel('Type de sources')

# plot title
#plt.title('Lisibilté')

# wrapping labels
ax = plt.axes()
def wrap_labels(ax, width, break_long_words=False):
    labels = []
    for label in ax.get_yticklabels():
        text = label.get_text()
        labels.append(textwrap.fill(text, width=width,
                      break_long_words=break_long_words))
    ax.set_yticklabels(labels, rotation=90, ma='center')
wrap_labels(ax, 10)

# save the plot as image
plt.savefig(entree_xml+'/sources_legibility.png')

# function to show the plot (needs to be after saving)
plt.show()
```

A.4 Correspondance genre, statut familial et marital

Voir le paragraphe.

```
# modules import
import pandas as pd
from google.colab import drive
#import numpy as np
#import matplotlib.pyplot as plt
#import matplotlib.ticker as mtick

# files path definition
drive.mount('/content/drive', force_remount=True)
entree_xml = "/content/drive/MyDrive/"

# set dataframe display options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

# source csv import
ushmm = pd.read_csv (entree_xml+'/personUSHMM.csv', encoding='utf8',sep=';'
                                        , index_col=False, dtype = str)
census = pd.read_csv (entree_xml+'/census.csv', encoding='utf8',sep=';',
                                        usecols = ['ushmm_id'],  dtype = str)
deportation = pd.read_csv (entree_xml+'/deportation.csv', encoding='utf8',
                                        sep=';', usecols = ['ushmm_id'],
                                        dtype = str)

# define irrelevant columns
drop_columns = ['Restricted','PersonType','Honorific','MiddleName','Suffix'
                                        , 'Description','pePlaceBirth-Place
                                        of Birth','peNationality-Nationality'
                                        ,'peMediaPrimary-Primary Media']

# filter individuals concerned by respectively the census and the
                                        deportation files
ushmm_census = ushmm[ushmm['PersonId'].isin(census['ushmm_id'])]
ushmm_deportation = ushmm[ushmm['PersonId'].isin(deportation['ushmm_id'])]

# control imported data
print(ushmm_census.info())
print(ushmm_deportation.info())

# group by, compute and label statictics (from ushmm_census or
```

```
                                        ushmm_deportation)
table = pd.DataFrame(ushmm_census.groupby(['Gender','peFamilyRelationship-
                                        Family Relationship','peMaritalStatus
                                        -Marital Status'], dropna=False).size
                                        (),columns=['Nombre']).reset_index()
table.rename(columns={'peFamilyRelationship-Family Relationship':'Family
                                        Relationship','peMaritalStatus-
                                        Marital Status':'Marital Status'},
                                        inplace = True)
table = table.sort_values(['Nombre'], ascending=[False])
print(table)
```

Census			
Gender	Family Relationship	Marital Status	Nombre
Female	Wife	NaN	2276
Female	Daughter	NaN	1830
Female	NaN	NaN	1124
Female	Child	NaN	1054
Female	Wife	Married	393
Female	Self	NaN	391
Female	Spouse	NaN	147
Female	Concubine	NaN	109
Female	Mother	NaN	56
Female	Concubine	Single	52
Female	Self	Widowed	51
Female	NaN	Widowed	29
Female	Sister	NaN	19
Female	Relative-in-law	NaN	16
Female	Self	Single	14
Female	Daughter	Single	13
Female	Self	Married	12
Female	Relative	NaN	6
Female	Concubine	Married	5
Female	NaN	Single	4
Female	Granddaughter	NaN	4
Female	Child	Married	2
Female	Other	NaN	2
Female	Wife	Single	2
Female	Grandchild	NaN	1
Female	Mother	Married	1

Female	Daughter	Married	1
Female	Other	Married	1
Female	Other	Single	1
Female	Grandmother	NaN	1
Male	Self	NaN	1957
Male	Self	Married	195
Male	NaN	NaN	167
Male	Child	NaN	161
Male	Husband	NaN	84
Male	Brother	NaN	20
Male	Self	Single	20
Male	Husband	Married	3
Male	Father	NaN	2
Male	Grandfather	NaN	1
NaN	Child	NaN	4893
NaN	Self	NaN	3206
NaN	NaN	NaN	2525
NaN	Self	Married	308
NaN	Relative	NaN	165
NaN	Self	Single	116
NaN	NaN	Single	28
NaN	NaN	Widowed	18
NaN	Grandchild	NaN	15
NaN	Self	Widowed	13
NaN	Child	Single	13
NaN	Spouse	NaN	9
NaN	NaN	Married	7
NaN	Relative-in-Law	NaN	4
NaN	Other	NaN	1
NaN	Relative-in-law	NaN	1

TABLE A.1 : Correspondance de trois attributs (Census)

Déportation			
Gender	Family Relationship	Marital Status	Nombre
Female	Daughter	NaN	1046
Female	Wife	Married	905
Female	NaN	NaN	522
Female	Child	NaN	374
Female	Self	NaN	279
Female	Self	Widowed	226
Female	Spouse	NaN	154
Female	Wife	NaN	142
Female	Concubine	Single	79
Female	Self	Married	58
Female	Mother	NaN	38
Female	Relative	NaN	36
Female	Granddaughter	NaN	32
Female	NaN	Married	26
Female	Relative-in-law	NaN	20
Female	NaN	Single	18
Female	Other	NaN	13
Female	Self	Single	13
Female	Sister	NaN	12
Female	NaN	Widowed	4
Female	Concubine	NaN	3
Female	Mother	Widowed	2
Female	Daughter	Married	2
Female	relative	NaN	2
Female	Concubine	Married	1
Female	Grandchild	NaN	1
Female	Relative	Widowed	1
Female	Child	Widowed	1
Female	Spouse	Married	1
Female	Spouse	Single	1
Female	Wife	Single	1
Female	Aunt	NaN	1
Male	Self	Married	789
Male	Self	NaN	454
Male	Child	NaN	281
Male	NaN	NaN	178

Male	Self	Single	61
Male	Brother	NaN	32
Male	Self	Widowed	26
Male	Son	NaN	7
Male	Father	NaN	3
Male	NaN	Single	1
Male	Grandfather	NaN	1
Male	Grandchild	NaN	1
NaN	Child	NaN	2886
NaN	NaN	NaN	2705
NaN	Self	NaN	1175
NaN	Self	Married	152
NaN	NaN	Single	112
NaN	Self	Single	41
NaN	Grandchild	NaN	39
NaN	Self	Widowed	27
NaN	Relative	NaN	16
NaN	NaN	Married	11
NaN	Other	NaN	9
NaN	Child	Single	5
NaN	Sibling	NaN	3
NaN	Relative-in-law	NaN	3
NaN	NaN	Widowed	1

TABLE A.2 : Correspondance de trois attributs (Déportation)

A.5 Correspondance genre, statut familial et marital

Voir le paragraphe.

```
# modules import
import pandas as pd
from google.colab import drive
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

# files path definition
drive.mount('/content/drive', force_remount=True)
entree_xml = "/content/drive/MyDrive/"

# set dataframe display options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

# source csv import (files previously prepared with the Prepare() function)
original_census_latest_prepared = pd.read_csv (entree_xml+'/
                                    original_census_latest_prepared.csv',
                                     encoding='utf8',sep=';', index_col=
                                    False, dtype = str)
original_deportation_latest_prepared = pd.read_csv (entree_xml+'/
                                    original_deportation_latest_prepared.
                                    csv', encoding='utf8',sep=';',
                                    index_col=False, dtype = str)

# remove irrelevant columns
original_census_latest_prepared = original_census_latest_prepared[['gender'
                                    ,'derived_gender']]
original_deportation_latest_prepared = original_deportation_latest_prepared
                                    [['gender','derived_gender']]

# renaming to prepare graph labels
original_census_latest_prepared.rename(columns = {'gender': 'Genre', '
                                    derived_gender': 'Genre inféré'},
                                    inplace = True)
original_deportation_latest_prepared.rename(columns = {'gender': 'Genre', '
                                    derived_gender': 'Genre inféré'},
                                    inplace = True)

# get columns coverage in %
```

```
cs_gender_percent_coverage = original_census_latest_prepared.notnull().sum
                                    () * 100 / len(
                                    original_census_latest_prepared)
dpt_gender_percent_coverage = original_deportation_latest_prepared.notnull
                                    ().sum() * 100 / len(
                                    original_deportation_latest_prepared)

# set % series as dataframes
census_gender_percent_coverage = pd.DataFrame(cs_gender_percent_coverage,
                                    columns=['Pourcentage']).reset_index
                                    ()
deportation_gender_percent_coverage = pd.DataFrame(
                                    dpt_gender_percent_coverage, columns=
                                    ['Pourcentage']).reset_index()

# insert source type to prepare pivot table
census_gender_percent_coverage.insert(1, 'Fichier', 'Census')
deportation_gender_percent_coverage.insert(1, 'Fichier', 'Déportation')

# rename columns to prepare pivot table
census_gender_percent_coverage.rename(columns = {'index': 'Attributs'},
                                    inplace = True)
deportation_gender_percent_coverage.rename( columns = {'index': 'Attributs'
                                    }, inplace = True)

# merge the 2 dataframes
scope_coverage = pd.concat([ census_gender_percent_coverage ,
                                    deportation_gender_percent_coverage])

# pivot table to create the graph
scope_content = scope_coverage.pivot_table( index = "Attributs", columns =
                                    "Fichier", values = "Pourcentage",
                                    aggfunc=np.sum , sort=False )

# set font size
plt.rcParams.update({'font.size': 16})

# plot % coverage by column in horizontal bars
scope_content.plot.barh(figsize=(12,6))

# moving legend
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left')

# naming the x-axis
plt.xlabel('Taux de couverture')

# naming the y-axis
```

```
plt.ylabel('Attributs du fichier')

# shortening labels (get text after '-' if any)
ax = plt.axes()
def wrap_labels(ax):
    labels = []
    for label in ax.get_yticklabels():
        text = label.get_text()
        labels.append(text.split('-')[1] if '-' in text else text)
    ax.set_yticklabels(labels, ma ='center')
wrap_labels(ax)

# desired format of the ticks, here '40%'
fmt = '%.0f%%'
# format the x axis
xticks = mtick.FormatStrFormatter(fmt)
ax.xaxis.set_major_formatter(xticks)

# reverse order y axis
ax.invert_yaxis()

# save the plot as image
plt.savefig(entree_xml+'/derived_gender.png', bbox_inches='tight')

# function to show the plot (needs to be after saving)
plt.show()
```

Annexe B

Codes développés au cours du projet

B.1 Standardisation

B.1.1 Format et encodage

Voir le paragraphe.

```
# modules import
import pandas as pd
from google.colab import drive
import numpy as np

# files path definition
drive.mount('/content/drive', force_remount=True)
entree_xml = "/content/drive/MyDrive/"

# set dataframe display options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

# transform newborns age to 0
def Convert_Age(x):
  if not x:
    return np.nan
  try:
    return '0' if '[' in str(x) else str(x)
  except:
    return np.nan

# load original data
```

```
ushmm = pd.read_csv (entree_xml+'/personUSHMM.csv', encoding='utf8',sep=';'
                                    , index_col=False, converters={'
                                    peNumberAge-Age': Convert_Age, '
                                    peNumberAge-Residence Age':
                                    Convert_Age}, dtype = str)

# load latest study files
individuals = pd.read_csv (entree_xml+'/individuals.csv', encoding='utf8',
                                    sep=';', usecols = ['id','gender','
                                    first_name','last_name','judet','
                                    household', 'relative', 'birthyear','
                                    marital_status'],  dtype = str)
census = pd.read_csv (entree_xml+'/census.csv', encoding='utf8',sep=';',
                                    usecols = ['individual','ushmm_id','
                                    line','digital_file','source','
                                    residence_territorial_entity'],
                                    dtype = str)
deportation = pd.read_csv (entree_xml+'/deportation.csv', encoding='utf8',
                                    sep=';', usecols = ['individual','
                                    ushmm_id','line','digital_file','
                                    source','residence_territorial_entity
                                    '], dtype = str)
census = pd.merge(individuals, census, left_on=['id'], right_on=['
                                    individual'], how='left').drop('
                                    individual', axis=1)
deportation = pd.merge(individuals, deportation, left_on=['id'], right_on=[
                                    'individual'], how='left').drop('
                                    individual', axis=1)

# filter records added manually
census = census[census['ushmm_id'].notna()]
census = census[~census['ushmm_id'].str.contains('n')]
deportation = deportation[deportation['ushmm_id'].notna()]
deportation = deportation[~deportation['ushmm_id'].str.contains('n')]

# save latest study files
census.to_csv(entree_xml+'census_latest.csv', encoding="utf_8_sig", sep = '
                                    ;', index=False)
deportation.to_csv(entree_xml+'deportation_latest.csv', encoding="utf_8_sig
                                    ", sep = ';', index=False)

# get relevant original data and merge (and remove modified study data)
mask_ushmm = ushmm[['PersonId','Gender','FirstName','LastName','peNumberAge
                                    -Age','peDateBirth-Date of Birth','
                                    peFamilyRelationship-Family
                                    Relationship','peMaritalStatus-
                                    Marital Status','pePlaceResidence-
```

```
                                        Residence']]
ushmm_census = pd.merge(census, mask_ushmm, left_on=['ushmm_id'], right_on=
                                        ['PersonId'], how='left').drop(['
                                        PersonId','gender','first_name','
                                        last_name','birthyear','relative','
                                        marital_status'], axis=1)
ushmm_deportation = pd.merge(deportation, mask_ushmm, left_on=['ushmm_id'],
                                         right_on=['PersonId'], how='left').
                                        drop(['PersonId','gender','first_name
                                        ','last_name','birthyear','relative',
                                        'marital_status'], axis=1)

# save latest study records with original data
ushmm_census.to_csv(entree_xml+'original_census_latest.csv', encoding="
                                        utf_8_sig", sep = ';', index=False)
ushmm_deportation.to_csv(entree_xml+'original_deportation_latest.csv',
                                        encoding="utf_8_sig", sep = ';',
                                        index=False)
```

B.1.2 Standardisation des colonnes

Voir le paragraphe.

```
# creating the "standard_cols_dict" dictionary used by Change_Column_Names(
                                    df_a,standard_cols_dict) to
                                    standardise column labels
  df_standard_cols = pd.read_excel (entree_xml+'/standard_columns2.xlsx',
                                      usecols=[0,1], dtype={'
                                      old_column_name':'object', '
                                      standard_column_name': 'object'})
  standard_cols_dict = pd.Series(df_standard_cols.standard_column_name.
                                      values,index=df_standard_cols.
                                      old_column_name).to_dict()

def Change_Column_Names(df, dict):
  """Function changing dataframe column labels to the standard names set
                                      out in the file standard_columns.
                                      xlsx
  :param df (dataframe): dataframe built from the source
  :param dict (dict): the dic created from the file standard_columns.xlsx
  :returns: df with standard column names that will be used in the code and
                                       outputs
  """
  use_cols = list(dict.keys())
  #print(use_cols)
  #print(df.columns)
  for y in df.columns:
    if y not in use_cols:
      #print('not in use_cols '+ y)
      df = df.drop(y, axis=1)
  df = df.rename(columns = dict)
  #print(df.info())
  return df
```

B.1.3 Standardisation des chaînes

Voir le paragraphe.

```
def Remove_Irrelevant_Char_Name(df, cols):
  """Function normalising input data (removing characters excluded by the
                                      Regex rules below), to enhance
                                      future matching and string
                                      comparisons
    :param df (dataframe): dataframe built from the source
    :param cols (columns, aka 'series' of dataframe to normalise): here,
                                        first & last names
    :returns: cleansed df for first & last names columns
  """
  irrelevant_regex = re.compile(r'[^\-\?\.a-zA-Z0-9\s]') # remove any
                                      character that is not - or ? or a
                                      dot followed by text+space
  multispace_regex = re.compile(r'\s\s+') # remove multiple spaces

  for col in cols:
    df[col]=df[col].str.replace(irrelevant_regex, ' ').str.replace(
                                        multispace_regex, ' ')
    df[col]=df[col].str.replace(re.compile(r'\.(?=[a-z])'), '. ') # dots
                                        immediately followed by dot are
                                        replaced by a space  #modifier
                                        pour les .. ...

  return df

  def Dot(df, var):
  """Function removing remaining dots, those following letters & followed
                                      by a space or end of string (dots
                                      used after abbreviations as in
                                      Nicolae N. Iancu)
    :param df (dataframe): dataframe built from the source
    :param var (columns, aka 'series' of dataframe to normalise): here, all
                                         segmented first & last names
    :returns: df with dot-cleansed first & last names columns
  """
  dot_regex = re.compile(r'(?<=[a-zA-Z])\.{1}?(?=\ |$ )') # identifies a
                                      unique dot immediately following
                                      letters and followed by space or
                                      end of string.
  for col in var:
    df[col]=df[col].str.replace(dot_regex,'')
  return df
```

B.1.4 Segmentation des prénoms et noms

Voir le paragraphe.

```
def Appellation(df, cols):
  """Function capturing and isolating nicknames intriduced by "zis"/"zisa"
                                        (Babu zis Arapu / ţNiu zis Vasile),
                                        to enhance future matching and
                                        string comparisons
    :param df (dataframe): dataframe built from the source
    :param cols (columns, aka 'series' of dataframe to normalise): here,
                                          first & last names
    :returns: cleansed df for first & last names columns
  """
  for col in cols:
    appellation = df[col].str.extract(r'( zis.*$)')
    df.insert(loc = df.columns.get_loc(col) + 1 , column = col.replace('
                                        name','') + '_appelation', value
                                        = appellation[0])
    df[col] = df[col].str.replace(r'( zis.*$)', '')
  return df

  def Split_Names(df, cols):
  """Function splitting first & last names, to enhance future matching and
                                       string comparisons
    :param df (dataframe): dataframe built from the source
    :param cols (columns, aka 'series' of dataframe to normalise): here,
                                          first & last names
    :returns: df with segmented first & last names columns
  """
  for col in cols:
    df_extraction = df[col].str.split(' ', expand=True)
    df_extraction.columns = [col+'_'+str(i+1) for i in df_extraction.
                                        columns]
    df[col] = df_extraction.iloc[:,0]
    x = list(df_extraction.columns).index(max(list(df_extraction.columns)))
    df = pd.concat([df, df_extraction.iloc[:,1:x+1]], axis=1)
    df = df.reindex(sorted(df.columns), axis=1)
    return df
```

B.1.5 Développement des abréviations de noms

Voir le paragraphe.

```
# creating the "df_spelling" dictionary used below to expand name
                                        abbreviations (C-tin to Constantin)
df_spelling = pd.read_excel (entree_xml+'/spelling.xlsx', usecols=[0,1],
                                        dtype={'Entry':'object', '
                                        Correct_Spelling': 'object'})
df_spelling = pd.Series(df_spelling.Correct_Spelling.values,index=
                                        df_spelling.Entry).to_dict()
df_spelling = {rf"\b{k}\b" : v for k, v in df_spelling.items()}
```

...

```
# expanding name abbreviations (C-tin to Constantin) from the df_spelling
                                        dictionary, built in preambule above
firstlast_cols=['firstname','lastname']
df_a.loc[:, firstlast_cols] = df_a[firstlast_cols].replace(df_spelling,
                                        regex=True)
df_b.loc[:, firstlast_cols] = df_b[firstlast_cols].replace(df_spelling,
                                        regex=True)
```

B.1.6 Enrichissements

B.1.6.1 Noms complets

Voir le paragraphe.

```
def Fullname(df, var):
  """Function removing remaining dots, those following letters & followed
                                        by a space or end of string (dots
                                        used after abbreviations as in
                                        Nicolae N. Iancu)
    :param df (dataframe): dataframe built from the source
    :param var (columns, aka 'series' of dataframe to normalise): here, all
                                          segmented first & last names
    :returns: df with dot-cleansed first & last names columns
  """
  last_column = var[-1]
  last_index = df.columns.get_loc(last_column)
  fullname = df[var].apply(lambda x: ' '.join(sorted(x.dropna().astype(str)
                                        )), axis=1)
  df.insert(loc = last_index + 1 , column = 'fullname', value = fullname )

  return df
```

B.1.6.2 Années de naissance

Voir le paragraphe.

```
def Compute_Birthyear(df, doc_year: int, file):
  """Function converting age, where available, to birth year
  :param df (dataframe): dataframe built from the source
  :param doc_year (YYYY-format integer): the date of the archive
  :param file (string): name of the file (without extension) for which we
                                        compute the birth year
  :returns: df with birthyear field computed from age if available
  """
  if 'birthyear' in df.columns and not 'age' in df.columns:
    print('The file '+ file + '.csv already has a birthyear field and no
                                          age field, formatting birthyear
                                          as str.')
    df['birthyear'] = df['birthyear'].apply(lambda x: str(int(x)) if not pd
                                          .isnull(x) else np.nan)
    return df
  elif (df.age.notnull() & df.birthyear.isnull()).sum() >=1:
    print('Calculating birthyear for :' + file)
    df['birthyear'] = df.apply(lambda row: int(doc_year) - int(row['age'])
                                          if (pd.isnull(row['birthyear'])
```

```
                                        and pd.notna(row['age'])) else
                                        row['birthyear'], axis=1)
    df['birthyear'] = df['birthyear'].apply(lambda x: str(int(x)) if not pd
                                        .isnull(x) else np.nan)
    df = df.astype(str)
  return df
```

B.1.6.3 Genre inféré

Voir le paragraphe.

```
def Derive_Gender(df):
  """Function deriving missing genders from the firstname-gender table
                                        gender.csv
  :param df (dataframe): dataframe built from the source
  :returns: df with derived genders from firstnames
  """
  # loading the gender mapping file
  firstnames_genders = pd.read_csv (entree_xml+'/gender.csv', encoding='
                                        utf8',sep=';', index_col=False,
                                        dtype = str)

  # filling blank values in original gender to 'undertermined'
  df['gender'] = df['gender'].fillna('undetermined')

  # create a mapping dictionary
  mapping = dict(firstnames_genders[['firstname', 'derived_gender']].values
                                        )

  # assign the mapped genders based on firstname in the source file and
                                        filling missing values as empty
  df['derived_gender'] = df['firstname'].map(mapping)
  df['derived_gender'].fillna(np.nan, inplace=True)

  # suppress derived genders not in alignment with relative status
  conditions = [(df['relative'].isin(['husband', 'brother','son','father','
                                        grandfather','frate','fratii','fiu'
                                        ]) & df['derived_gender'].eq('
                                        female')),
                (df['relative'].isin(['wife', 'spouse','daughter','
                                                    concubine','mother','
                                                    sister','
                                                    granddaughter','
                                                    grandmother','aunt','
                                                    fiica','sora']) & df[
                                                    'derived_gender'].eq(
                                                    'male'))]
  choices = ['undetermined','undetermined']

  # where derived genders != relative status, set to undetermined,
                                        otherwise leave derived value
  df['derived_gender'] = np.select(conditions, choices, df['derived_gender'
                                        ])

  # where derived genders empty, fill with original genders (also set '
```

```
                                        undetermined' to NaN where
                                        applicable)
df['derived_gender'] = df['derived_gender'].fillna(df['gender'])

# resetting type as str
df[['gender','derived_gender']] = df[['gender','derived_gender']].astype(
                                        str)

# congruency control between original genders and derived genders
#print('\n "gender derived = gender" congruence : \n', df.groupby(['
                                        gender', 'derived_gender'])['
                                        derived_gender'].count())

return df
```

B.2 Appariements

B.2.1 Scores de similarité

Voir le paragraphe.

```
def Test_Scores(df):
  """Function calculating a set of string similarities between 2 files
                                        already matched
    :param df (string): name without extension of the ids mapping (ushmm_id
                                         ) file between the 2 matched
                                         files
    :returns: df with a set of string similarities between 2 files already
                                         matched
  """
  # load the persons ids mapping file
  manual_matches = pd.read_csv (entree_xml + '/' + df + '.csv', encoding='
                                        utf8',sep=';', index_col=False,
                                        dtype = str)

  # load the files
  original_census_latest_prepared = pd.read_csv (entree_xml + '/
                                        original_census_latest_prepared.csv
                                        ', encoding='utf8',sep=';',
                                        index_col=False, dtype = str)
  original_deportation_latest_prepared = pd.read_csv (entree_xml + '/
                                        original_deportation_latest_prepared
                                        .csv', encoding='utf8',sep=';',
                                        index_col=False, dtype = str)

  # merge prepared original data based on the mapping ids file
  scores = pd.merge(manual_matches, original_census_latest_prepared,
                                        left_on=['census'], right_on=['
                                        ushmm_id'], how='left').drop('
                                        census', axis=1)
  scores = pd.merge(scores, original_deportation_latest_prepared, left_on=[
                                        'deportation'], right_on=['ushmm_id
                                        '], how='left').drop('deportation',
                                         axis=1)

  # select rows where data is available and persons not having firstname =
                                        lastname (to evaluate matches where
                                         first and last names are reversed)
  names_notna = scores[scores['firstname_x'].notnull() & scores['lastname_y
                                        '].notnull() & scores['firstname_y'
                                        ].notnull() & scores['lastname_x'].
                                        notnull() & (scores['firstname_x']
                                        not in scores['lastname_x'])][['
```

```
                                    ushmm_id_x','firstname_x','
                                    lastname_x','firstname_y','
                                    lastname_y']]

# create reversed first/last names indicator based on 80% levenshstein
                                    match
names_notna['reversed'] =  names_notna[['firstname_x','lastname_x','
                                    firstname_y','lastname_y']].apply(
                                    lambda x: 'Y' if (txtlev(x[0],x[3])
                                     >= 0.8 and txtlev(x[2],x[1]) >= 0.
                                    8) else 'N', axis=1)

# only keep ids for the selection and change mergekey name (to remove it
                                    in next step)
names_notna.drop(['firstname_x','lastname_x','firstname_y','lastname_y'],
                                     axis=1, inplace=True)
names_notna.rename({'ushmm_id_x': 'temp_id'}, axis=1, inplace=True)

# append reversed indicator to the main score file, fillna with indicator
                                     N (names not reversed)
scores = pd.merge(scores, names_notna, left_on=['ushmm_id_x'], right_on=[
                                    'temp_id'], how='left').drop('
                                    temp_id', axis=1)
scores['reversed'] = scores['reversed'].fillna('N')

# get best match between individuals firstnames

scores['firstnames_fuzzy_tok_set'] = scores[['original_first_x','
                                    original_first_y']].apply(lambda x:
                                     combinations(x[0],x[1])[0:2] if(np
                                    .all(pd.notnull(x[0])) and np.all(
                                    pd.notnull(x[1]))) else None, axis=
                                    1) # if (' ' in x[0] or ' ' in x[1
                                    ]) else None
scores['best_firstname_x'] = scores['firstnames_fuzzy_tok_set'].apply(
                                    lambda x: x[0] if x else None)
scores['best_firstname_y'] = scores['firstnames_fuzzy_tok_set'].apply(
                                    lambda x: x[-1] if x else None) #
                                    isinstance(x, list)

# calculate set of scores on available data (field by field : atomic
                                    comparison)

scores['Additional_FirstNames_x'] = scores[['best_firstname_x','
                                    original_first_x']].apply(lambda x:
                                     ' '.join([str(i) for i in x[1].
                                    split() if str(i) != x[0]]) if(np.
```

```
                                        all(pd.notnull(x[0])) and np.all(pd
                                        .notnull(x[1]))) else None, axis=1)
scores['Additional_FirstNames_y'] = scores[['best_firstname_y','
                                        original_first_y']].apply(lambda x:
                                         ' '.join([str(i) for i in x[1].
                                        split() if str(i) != x[0]]) if(np.
                                        all(pd.notnull(x[0])) and np.all(pd
                                        .notnull(x[1]))) else None, axis=1)

# scores['sc_Family_Firstnames'] = scores[['fam_firstnames_x','
                                        fam_firstnames_y']].apply(lambda x:
                                         round(pyMEJW((' '.join(map(str, x[
                                        0]))).split(),(' '.join(map(str, x[
                                        1]))).split()),2) if(np.all(pd.
                                        notnull(x[0])) and np.all(pd.
                                        notnull(x[1]))) else None, axis=1)

scores['sc_best_firstname_lev'] = scores[['best_firstname_x','
                                        best_firstname_y']].apply(lambda x:
                                         round(txtlev(x[0],x[1]),2) if(np.
                                        all(pd.notnull(x[0])) and np.all(pd
                                        .notnull(x[1]))) else None, axis=1)
scores['sc_firstname_lev'] = scores[['firstname_x','firstname_y']].apply(
                                        lambda x: round(txtlev(x[0],x[1]),2
                                        ) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)
scores['sc_lastname_lev'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtlev(x[0],x[1]),2
                                        ) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)

scores['sc_firstname_dlev'] = scores[['firstname_x','firstname_y']].apply
                                        (lambda x: round(txtdlev(x[0],x[1])
                                        ,2) if(np.all(pd.notnull(x[0])) and
                                         np.all(pd.notnull(x[1]))) else
                                        None, axis=1)
scores['sc_lastname_dlev'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtdlev(x[0],x[1]),
                                        2) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)

scores['sc_firstname_jw'] = scores[['firstname_x','firstname_y']].apply(
                                        lambda x: round(txtjw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
```

```
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
scores['sc_lastname_jw'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtjw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)

scores['sc_firstname_sw'] = scores[['firstname_x','firstname_y']].apply(
                                        lambda x: round(txtsw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
scores['sc_lastname_sw'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtsw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)

scores['sc_fullname_lev'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtlev(x[0],x[1]),2
                                        ) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)
scores['sc_fullname_dlev'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtdlev(x[0],x[1]),
                                        2) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)
scores['sc_fullname_jw'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtjw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
scores['sc_fullname_sw'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtsw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
# calculating Monge-Elkan only for fullname as it has the property of
                                        processing multiword comparisons
scores['sc_fullname_ME'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(pyMEJW(x[0].split()
                                        ,x[1].split()),2) if(np.all(pd.
                                        notnull(x[0])) and np.all(pd.
                                        notnull(x[1]))) else None, axis=1)
```

```
# calculating mean of atomic scores between first and lastnames results (
                                    except Monge-Elkan as indicated)
scores['sc_lev_avg'] = round(scores[['sc_firstname_lev', 'sc_lastname_lev
                                    ']].mean(axis=1),2)
scores['sc_dlev_avg'] = round(scores[['sc_firstname_dlev', '
                                    sc_lastname_dlev']].mean(axis=1),2)
scores['sc_jw_avg'] = round(scores[['sc_firstname_jw', 'sc_lastname_jw']]
                                    .mean(axis=1),2)
scores['sc_sw_avg'] = round(scores[['sc_firstname_sw', 'sc_lastname_sw']]
                                    .mean(axis=1),2)


# converting to float
float_scores_cols = [col for col in scores if col.startswith('sc_')]
scores[float_scores_cols] = scores[float_scores_cols].astype(float)

# save latest study records (original data) prepared & scored
scores.to_csv(entree_xml + '/scores_test.csv', encoding="utf_8_sig", sep
                                    = ';', decimal=",", index=False)

return scores
```

B.2.2 Scores de similarité

Voir le paragraphe.

```
def Record_Linkage(file_a, file_b):
  """Function performing Record Linkage Matching between (provided file_a).
                                        csv & (provided file_b).csv,
                                        returning the set of results for
                                        analysis
    :param file_a (string): name without extension of the first file to be
                                          used in Record Linkage
    :param file_b (string): name without extension of the second file to be
                                           used in Record Linkage
    :returns:
  """
  # test whether the prepared files exist
  try:
    df_a = pd.read_csv (entree_xml+'/'+ file_a + '.csv', encoding='utf8',
                                          sep=';', index_col=False, dtype =
                                           str)
  except:
    return print('The file '+ file_a + 'does not exist, please prepare the
                                          file first and retry')
  try:
    df_b = pd.read_csv (entree_xml+'/'+ file_b + '.csv', encoding='utf8',
                                          sep=';', index_col=False, dtype =
                                           str)
  except:
    return print('The file '+ file_b + 'does not exist, please prepare the
                                          file first and retry')


  # defining blockkeys
  df_a['blockkey1'] = df_a['residence_territorial_entity']
  df_b['blockkey1'] = df_b['residence_territorial_entity']

  # other examples and ways of creating blocking key for the user
                                        information
  # defining combinations of fields
  # blockkey2_cols = ['residence_territorial_entity','birthyear','firstname
                                        ','lastname']
  # creating the above fields combination
  # df_a['blockkey2'] = df_a[blockkey2_cols].fillna('').sum(axis=1)

  # definign a blockkey with the first 2 caracters of the territorial
                                        entity and the initial of the
                                        firstname and last caracter of
                                        firstname
```

```
# df_a['blockkey3'] = df_a['residence_territorial_entity'].str[:2] + df_a
                                        ['firstname'].str[:1] + df_a['
                                        firstname'].str[-1]

# set up indexer for Recordlinkage
indexer = recordlinkage.Index()
#indexer.full() # full index, this exceeds Colab's memory capacity
indexer.block(['blockkey1','derived_gender'],['blockkey1','derived_gender
                                        '])
#indexer.block('digital_file','digital_file')
#indexer.block('lastname','lastname')
#indexer.block('derived_gender','derived_gender') # this alone exceeds
                                        Colab's memory capacity
#indexer.block('relative','relative')
#indexer.block('birthyear','birthyear')
#indexer.add(SortedNeighbourhood(left_on='lastname', right_on='lastname',
                                         block_on=['blockkey1','
                                        derived_gender'], window=9))

compare_cols = ['ushmm_id', 'birthyear', 'firstname', 'firstname_2', '
                                        firstname_3', 'derived_gender', '
                                        lastname', 'fullname', 'relative',
                                        'household','
                                        residence_territorial_entity', '
                                        source_id', 'blockkey1']
df_a = df_a.filter(compare_cols)
df_b = df_b.filter(compare_cols)

#df_a.index = np.arange(len(df_a))
#df_b.index = np.arange(len(df_b))

df_a.set_index('ushmm_id', inplace=True, drop=True)
df_b = df_b.set_index('ushmm_id')

print('\n dfa_tomatch head\n', df_a.head(2))
print('\n dfb_tomatch head\n', df_b.head(2))

# index pairs
print('\n dfa cols :\n', df_a.columns)
print('\n dfb cols :\n', df_b.columns)

candidate_pairs = indexer.index(df_a, df_b)
print("candidate_pairs:", len(candidate_pairs))
print('\ncandidate_pairs index : \n',candidate_pairs)


# initialise class
```

```
comp = recordlinkage.Compare()

# initialise similarity measurement algorithms
comp.string('firstname', 'firstname', method='levenshtein', threshold=0.
                                      70, missing_value=0, label='
                                      firstname')
comp.string('firstname_2', 'firstname_2', method='levenshtein', threshold
                                      =0.70, missing_value=0, label='
                                      firstname2')
comp.string('lastname', 'lastname', method='levenshtein', threshold=0.80,
                                       missing_value=0, label='lastname')
comp.string('fullname', 'fullname', method='levenshtein', threshold=0.80,
                                       missing_value=0, label='fullname')
#comp.string('firstname', 'lastname', method='levenshtein', threshold=0.
                                      85, label='firstname in last')
#comp.string('lastname', 'firstname', method='levenshtein', threshold=1,
                                      label='lastname in first')
comp.exact('birthyear', 'birthyear',  missing_value=0, label='birthyear')
#comp.numeric('birthyear', 'birthyear', method='linear', offset=3, scale=
                                      3, missing_value=0.5, label='
                                      birthyear') #too much ressources
                                      needed, session crash
comp.exact('household', 'household',  missing_value=0, label='household')
comp.exact('derived_gender', 'derived_gender',  missing_value=0, label='
                                      gender')
comp.exact('relative', 'relative', missing_value=0, label='relative')
comp.exact('residence_territorial_entity', 'residence_territorial_entity'
                                      , label='
                                      residence_territorial_entity')

# the method .compute() returns the DataFrame with the feature vectors (1
                                       for fields where the rule is
                                      fullfiled, 0 otherwise).
features = comp.compute(candidate_pairs, df_a, df_b)

# displaying results statistics
# mean, standard, quantile, etc of results
print(features.describe())
# sum of pairs by number of attributes matching the rules
print(features.sum(axis=1).value_counts().sort_index(ascending=False))
# control view on screen
print(features.head(10))

# filtering potential matches where rules fullfilled total >=3 (can be
                                      adjusted) and both first &
                                      lastnames are matching the above
                                      criteria
```

```
potential_matches = features[(features.sum(axis=1) >=3) & (features[['
                                    firstname','lastname']].sum(axis=1)
                                     >1)].reset_index()
# add a column summing all results of the comparisons
potential_matches['Score'] = potential_matches.loc[:, 'firstname':'
                                    residence_territorial_entity'].sum(
                                    axis=1)
# control view on screen
print(potential_matches.head(10))

# RL toolkit only returns index and results, original data needs to be
                                    mapped back based on id
# select columns to display with the results (usually the ones used in
                                    comparison rules)
df_a_lookup = df_a[['fullname','firstname_2','birthyear','derived_gender'
                                    ,'relative','household','
                                    residence_territorial_entity','
                                    source_id']].reset_index()
df_b_lookup = df_b[['fullname','firstname_2','birthyear','derived_gender'
                                    ,'relative','household','
                                    residence_territorial_entity','
                                    source_id']].reset_index()

# adding the above columns to the results
df_a_merge = pd.merge(potential_matches, df_a_lookup, left_on='ushmm_id_1
                                    ',right_on='ushmm_id').drop('
                                    ushmm_id', axis=1)
final_merge = pd.merge(df_a_merge, df_b_lookup, left_on='ushmm_id_2',
                                    right_on='ushmm_id').drop('ushmm_id
                                    ', axis=1)
# control view on screen
print(final_merge.head(10))

# exporting results to csv
final_merge.to_csv(entree_xml+'/Record_Linkage.csv', encoding="utf_8_sig"
                                    , sep = ';', decimal=",", index=
                                    False)

#########
# Unsupervised maching and evaluation of performance
#########

feature_vectors = comp.compute(candidate_pairs, df_a, df_b)

# create training and test sets (a model should not be trained and tested
                                     on the same subset of the data)
train, test = train_test_split(feature_vectors, test_size=0.25)
```

```
# load the reference matching pairs
manual_matches = pd.read_csv (entree_xml+'/
                                    census_deportation_ground_truth.csv
                                    ', encoding='utf8',sep=';',
                                    index_col=False, dtype = str)
manual_matches = manual_matches.set_index(['ushmm_id_x', 'ushmm_id_y']).
                                    index

# load true pairs for the evalation of the test
test_matches_index = test.index & manual_matches

# built the K-mean classifier
kmeans = recordlinkage.KMeansClassifier()

# command for the model training
result_kmeans = kmeans.learn(train)

# build the predictions on the test set
predictions = kmeans.predict(test)

# prepare the confusion matrix (True/False positives and True/False
                                    negatives)
confusion_matrix = recordlinkage.confusion_matrix(test_matches_index,
                                    predictions, len(test))

# display the precision, recall and F-measure scores
print('Précision (Precision) : ', recordlinkage.precision(
                                    confusion_matrix))
print('Rappel (Recall) : ', recordlinkage.recall(confusion_matrix))
print('F1-Score (F-Measure) : ', recordlinkage.fscore(confusion_matrix))

return final_merge
```

B.2.3 Scores de similarité

```
'''
Projet réalisé par JV BOBY dans le cadre du stage de fin d'étude
Mémoire pour le diplôme de master « Technologies numériques appliquées à
                                        'lhistoire »
2022

La déportation des Roms en Transnistrie, 1942-1944.
Étude de 'lappariement de listes de déportés.

Réalisé avec Python Record Linkage Toolkit:
A toolkit for record linkage and duplicate detection in Python
https://github.com/J535D165/recordlinkage

'''

# modules import
import pandas as pd
from google.colab import drive
from random import randrange
#!pip install Levenshtein
#import Levenshtein as lev
#from Levenshtein import *
!pip install fuzzywuzzy
from fuzzywuzzy import fuzz, process
import itertools
from operator import itemgetter
import unicodedata
import numpy as np
import re
!pip install "textdistance[extras]"
import textdistance

!pip install py_stringmatching
from py_stringmatching import MongeElkan as ME
from py_stringmatching import JaroWinkler as JW

# prepare alias for the textdistance modules
txtlev = textdistance.levenshtein.normalized_similarity
txtdlev = textdistance.damerau_levenshtein.normalized_similarity
txtjw = textdistance.jaro_winkler.normalized_similarity
txtedtx = textdistance.editex.normalized_similarity
txtsw = textdistance.smith_waterman.normalized_similarity
txtjccd = textdistance.jaccard.normalized_similarity
pyMEJW = ME(sim_func=JW().get_sim_score).get_raw_score
```

```
!pip install recordlinkage
import recordlinkage
from recordlinkage.index import SortedNeighbourhood
from recordlinkage.index import Block

from sklearn.model_selection import train_test_split

# files path definition
drive.mount('/content/drive', force_remount=True)
entree_xml = "/content/drive/MyDrive/"

# set dataframe display options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

# creating the "df_spelling" dictionary used below to expand name
                                    abbreviations (C-tin to Constantin)
df_spelling = pd.read_excel (entree_xml+'/spelling.xlsx', usecols=[0,1],
                                    dtype={'Entry':'object', '
                                    Correct_Spelling': 'object'})
df_spelling = pd.Series(df_spelling.Correct_Spelling.values,index=
                                    df_spelling.Entry).to_dict()
df_spelling = {rf"\b{k}\b" : v for k, v in df_spelling.items()}

# transform newborns age to 0
def Convert_Age(x):
  if not x:
    return np.nan
  try:
    return '0' if '[' in str(x) else str(x)
  except:
    return np.nan

# load original data
ushmm = pd.read_csv (entree_xml+'/personUSHMM.csv', encoding='utf8',sep=';'
                                    , index_col=False, converters={'
                                    peNumberAge-Age': Convert_Age, '
                                    peNumberAge-Residence Age':
                                    Convert_Age}, dtype = str)

# load latest study files
individuals = pd.read_csv (entree_xml+'/individuals.csv', encoding='utf8',
                                    sep=';', usecols = ['id','gender','
```

```
                                        first_name','last_name','judet','
                                        household', 'relative', 'birthyear','
                                        marital_status'],  dtype = str)
census = pd.read_csv (entree_xml+'/census.csv', encoding='utf8',sep=';',
                                        usecols = ['individual','ushmm_id','
                                        line','digital_file','source','
                                        residence_territorial_entity'],
                                        dtype = str)
deportation = pd.read_csv (entree_xml+'/deportation.csv', encoding='utf8',
                                        sep=';', usecols = ['individual','
                                        ushmm_id','line','digital_file','
                                        source','residence_territorial_entity
                                        '], dtype = str)
census = pd.merge(individuals, census, left_on=['id'], right_on=['
                                        individual'], how='left').drop('
                                        individual', axis=1)
deportation = pd.merge(individuals, deportation, left_on=['id'], right_on=[
                                        'individual'], how='left').drop('
                                        individual', axis=1)

# filter records added manually
census = census[census['ushmm_id'].notna()]
census = census[~census['ushmm_id'].str.contains('n')]
deportation = deportation[deportation['ushmm_id'].notna()]
deportation = deportation[~deportation['ushmm_id'].str.contains('n')]

# sort dataframes in individual appearing order (by id)
census = census.sort_values(by=['ushmm_id'])
deportation = deportation.sort_values(by=['ushmm_id'])

# save latest study files
census.to_csv(entree_xml+'census_latest.csv', encoding="utf_8_sig", sep = '
                                        ;', index=False)
deportation.to_csv(entree_xml+'deportation_latest.csv', encoding="utf_8_sig
                                        ", sep = ';', index=False)

# get relevant original data and merge, suppress study equivalents
mask_ushmm = ushmm[['PersonId','Gender','FirstName','LastName','peNumberAge
                                        -Age','peDateBirth-Date of Birth','
                                        peFamilyRelationship-Family
                                        Relationship','peMaritalStatus-
                                        Marital Status','pePlaceResidence-
                                        Residence']]
ushmm_census = pd.merge(census, mask_ushmm, left_on=['ushmm_id'], right_on=
                                        ['PersonId'], how='left').drop(['
                                        PersonId','gender','first_name','
                                        last_name','birthyear','relative','
```

```
                                        marital_status'], axis=1)
ushmm_deportation = pd.merge(deportation, mask_ushmm, left_on=['ushmm_id'],
                                         right_on=['PersonId'], how='left').
                                        drop(['PersonId','gender','first_name
                                        ','last_name','birthyear','relative',
                                        'marital_status'], axis=1)

# sort dataframes in individual appearing order (by id)
ushmm_census = ushmm_census.sort_values(by=['ushmm_id'])
ushmm_deportation = ushmm_deportation.sort_values(by=['ushmm_id'])

# save latest study records with original data
ushmm_census.to_csv(entree_xml+'original_census_latest.csv', encoding="
                                        utf_8_sig", sep = ';', index=False)
ushmm_deportation.to_csv(entree_xml+'original_deportation_latest.csv',
                                        encoding="utf_8_sig", sep = ';',
                                        index=False)

def getlatestindividuals():

  individuals = pd.read_csv (entree_xml+'/individuals.csv', encoding='utf8'
                                         ,sep=';', usecols = ['id','gender',
                                         'first_name','last_name','judet','
                                         household', 'relative', 'birthyear'
                                         ],  dtype = str)
  census = pd.read_csv (entree_xml+'/census.csv', encoding='utf8',sep=';',
                                         usecols = ['individual','ushmm_id',
                                         'line','digital_file','source','
                                         residence_territorial_entity'],
                                         dtype = str)
  deportation = pd.read_csv (entree_xml+'/deportation.csv', encoding='utf8'
                                         ,sep=';', usecols = ['individual','
                                         ushmm_id','line','digital_file','
                                         source','
                                         residence_territorial_entity'],
                                         dtype = str)
  individuals_latest = pd.merge(individuals, census, left_on=['id'],
                                         right_on=['individual'], how='left'
                                         ).drop('individual', axis=1)

  individuals_latest['source_type'] = np.where(individuals_latest['ushmm_id
                                         '].notnull(), 'census', np.nan)
  deportation.rename({'individual': 'id'}, axis=1, inplace=True)

  mapping = deportation.set_index('id')
```

```
  for c in ['ushmm_id','line','digital_file','source','
                                        residence_territorial_entity']:
    individuals_latest[c] = individuals_latest[c].fillna(individuals_latest
                                          ['id'].map(mapping[c]))

  #individuals_latest.loc[individuals_latest['ushmm_id'].notna() &
                                        individuals_latest['source_type'].
                                        isnull(), 'source_type'] = '
                                        deportation'
  individuals_latest['source_type'] = np.where((individuals_latest['
                                        ushmm_id'].notnull()) & (
                                        individuals_latest['source_type'] !
                                        = 'census'), 'deportation',
                                        individuals_latest['source_type'])

  individuals_latest.to_csv(entree_xml+'individuals_latest.csv', encoding="
                                        utf_8_sig", sep = ';', index=False)

  return print('individuals_latest.csv ready')


def Change_Column_Names(df, dict):
  """Function changing dataframe column labels to the standard names set
                                        out in the file standard_columns.
                                        xlsx
  :param df (dataframe): dataframe built from the source
  :param dict (dict): the dic created from the file standard_columns.xlsx
  :returns: df with standard column names that will be used in the code and
                                         outputs
  """
  use_cols = list(dict.keys())
  #print(use_cols)
  #print(df.columns)
  for y in df.columns:
    if y not in use_cols:
      #print('not in use_cols '+ y)
      df = df.drop(y, axis=1)
  df = df.rename(columns = dict)
  #print(df.info())
  return df

def Compute_Birthyear(df, doc_year: int, file):
  """Function converting age, where available, to birth year
  :param df (dataframe): dataframe built from the source
  :param doc_year (YYYY-format integer): the date of the archive
  :param file (string): name of the file (without extension) for which we
                                        compute the birth year
```

```
    :returns: df with birthyear field computed from age if available
    """
    if 'birthyear' in df.columns and not 'age' in df.columns:
        print('The file '+ file + '.csv already has a birthyear field and no
                                                age field, formatting birthyear
                                                as str.')
        df['birthyear'] = df['birthyear'].apply(lambda x: str(int(x)) if not pd
                                                .isnull(x) else np.nan)
        return df
    elif (df.age.notnull() & df.birthyear.isnull()).sum() >=1:
        print('Calculating birthyear for :' + file)
        df['birthyear'] = df.apply(lambda row: int(doc_year) - int(row['age'])
                                                if (pd.isnull(row['birthyear'])
                                                and pd.notna(row['age'])) else
                                                row['birthyear'], axis=1)
        df['birthyear'] = df['birthyear'].apply(lambda x: str(int(x)) if not pd
                                                .isnull(x) else np.nan)
        df = df.astype(str)
    return df

def Normalise(df):
    """Function normalising input data (lowercase, removing diacritics), to
                                            enhance future matching and string
                                            comparisons
        :param df (dataframe): dataframe built from the source
        :returns: lowercase, diacritic-free standardised dataframe
    """
    cols = df.select_dtypes(include=[object]).columns
    df[cols] = df[cols].apply(lambda x: x.str.lower().str.normalize('NFKD').
                                            str.encode('ascii', errors='ignore'
                                            ).str.decode('utf-8'), axis=0)
    return df

def Appellation(df, cols):
    """Function capturing and isolating nicknames intriduced by "zis"/"zisa"
                                            (Babu zis Arapu / ţNiu zis Vasile),
                                            to enhance future matching and
                                            string comparisons
        :param df (dataframe): dataframe built from the source
        :param cols (columns, aka 'series' of dataframe to normalise): here,
                                                first & last names
        :returns: cleansed df for first & last names columns
    """
    for col in cols:
        appellation = df[col].str.extract(r'( zis.*$)')
        df.insert(loc = df.columns.get_loc(col) + 1 , column = col.replace('
                                            name','') + '_appelation', value
```

```
                                        = appellation[0])
    df[col] = df[col].str.replace(r'( zis.*$)', '')
  return df

def Remove_Irrelevant_Char_Name(df, cols):
  """Function normalising input data (removing characters excluded by the
                                      Regex rules below), to enhance
                                      future matching and string
                                      comparisons
    :param df (dataframe): dataframe built from the source
    :param cols (columns, aka 'series' of dataframe to normalise): here,
                                        first & last names
    :returns: cleansed df for first & last names columns
  """
  irrelevant_regex = re.compile(r'[^\-\?\.a-zA-Z0-9\s]') # remove any
                                      character that is not - or ? or a
                                      dot followed by text+space
  multispace_regex = re.compile(r'\s\s+') # remove multiple spaces

  for col in cols:
    df[col]=df[col].str.replace(irrelevant_regex, ' ').str.replace(
                                        multispace_regex, ' ')
    df[col]=df[col].str.replace(re.compile(r'\.(?=[a-z])'), '. ') # dots
                                        immediately followed by dot are
                                        replaced by a space  #modifier
                                        pour les .. ...

  return df

def Dot(df, var):
  """Function removing remaining dots, those following letters & followed
                                      by a space or end of string (dots
                                      used after abbreviations as in
                                      Nicolae N. Iancu)
    :param df (dataframe): dataframe built from the source
    :param var (columns, aka 'series' of dataframe to normalise): here, all
                                         segmented first & last names
    :returns: df with dot-cleansed first & last names columns
  """
  dot_regex = re.compile(r'(?<=[a-zA-Z])\.{1}?(?=\ |\$)') # identifies a
                                      unique dot immediately following
                                      letters and followed by space or
                                      end of string.
  for col in var:
    df[col]=df[col].str.replace(dot_regex,'')
  return df
```

```
def List_Firstnames(df):
  """Function listing an individual firstnames
    :param df (dataframe): dataframe built from the source
    :returns: df with 2 new fields, one storing list of firstnames, another
                                          the number of these firstnames
  """
  df['list_firstnames'] = df['firstname'].apply(lambda x: x.split(' ') if
                                         np.all(pd.notnull(x)) else np.nan)
  df['firstnames_count'] = df['firstname'].apply(lambda x: len(str(x).split
                                         (' ')) if np.all(pd.notnull(x))
                                         else 0)
  return df

def combinations(left, right):
  """Function looping throught available firstnames for an individual and
                                         returning the best match
    :param left (string): list of firstnames from the left file of the
                                           match
    :param right (string): list of firstnames from the right file of the
                                           match
    :returns: best firstname match as a string
  """
  matches = []
  best = []
  if left and right:
    leftlist = list(map(str, left.strip().split(" ")))
    rightlist = list(map(str, right.split(" ")))
    for seq_1 in leftlist:
      for seq_2 in rightlist:
        if len(seq_1)>1 and len(seq_2)>1:
          if seq_1 == seq_2:
            matches.append((seq_1, seq_2, 1))
            leftlist.remove(seq_1)
            rightlist.remove(seq_2)
          else:
            matches.append((seq_1, seq_2, round(txtlev(seq_1, seq_2),2)))
        elif (len(seq_1) * len(seq_2))>1:
          if seq_2.startswith(seq_1) or seq_1.startswith(seq_2):
            matches.append((seq_1, seq_2, 0.5))
    if matches:
      best = [max(matches,key=itemgetter(2))[0], max(matches,key=itemgetter
                                           (2))[1],max(matches,key=
                                           itemgetter(2))[2]]
    else:
      best= ['None','None',0]
  else:
```

```
    best= ['None','None',0]
  return best


def Split_Names(df, cols):
  """Function splitting first & last names, to enhance future matching and
                                        string comparisons
    :param df (dataframe): dataframe built from the source
    :param cols (columns, aka 'series' of dataframe to normalise): here,
                                          first & last names
    :returns: df with segmented first & last names columns
  """
  # keep copy of original firstname (needed for other functions)
  df['original_first'] = df['firstname']
  # split names in incremental columns firstname_1, firstname_2,
                                        firstname_i (i last firstname index
                                        )
  for col in cols:
    df_extraction = df[col].str.split(' ', expand=True)
    df_extraction.columns = [col+'_'+str(i+1) for i in df_extraction.
                                          columns]
    df[col] = df_extraction.iloc[:,0]
    x = list(df_extraction.columns).index(max(list(df_extraction.columns)))
    df = pd.concat([df, df_extraction.iloc[:,1:x+1]], axis=1)
    df = df.reindex(sorted(df.columns), axis=1)
    return df

#ex dot place


def Fullname(df, var):
  """Function removing remaining dots, those following letters & followed
                                        by a space or end of string (dots
                                        used after abbreviations as in
                                        Nicolae N. Iancu)
    :param df (dataframe): dataframe built from the source
    :param var (columns, aka 'series' of dataframe to normalise): here, all
                                          segmented first & last names
    :returns: df with dot-cleansed first & last names columns
  """
  last_column = var[-1]
  last_index = df.columns.get_loc(last_column)
  fullname = df[var].apply(lambda x: ' '.join(sorted(x.dropna().astype(str)
                                        )), axis=1)
  df.insert(loc = last_index + 1 , column = 'fullname', value = fullname )

  return df
```

```
def Derive_Gender(df):
  """Function deriving missing genders from the firstname-gender table
                                        gender.csv
  :param df (dataframe): dataframe built from the source
  :returns: df with derived genders from firstnames
  """
  # loading the gender mapping file
  firstnames_genders = pd.read_csv (entree_xml+'/gender.csv', encoding='
                                        utf8',sep=';', index_col=False,
                                        dtype = str)

  # filling blank values in original gender to 'undertermined'
  df['gender'] = df['gender'].fillna('undetermined')

  df['first_firstname'] = df['firstname'].apply(lambda x: x.split(' ')[0]
                                        if np.all(pd.notnull(x)) else np.
                                        nan)
  # create a mapping dictionary
  mapping = dict(firstnames_genders[['firstname', 'derived_gender']].values
                                        )

  # assign the mapped genders based on firstname in the source file and
                                        filling missing values as empty
  df['derived_gender'] = df['first_firstname'].map(mapping)
  df['derived_gender'].fillna(np.nan, inplace=True)

  # suppress derived genders not in alignment with relative status
  conditions = [(df['relative'].isin(['husband', 'brother','son','father','
                                        grandfather','frate','fratii','fiu'
                                        ]) & df['derived_gender'].eq('
                                        female')),
                (df['relative'].isin(['wife', 'spouse','daughter','
                                                    concubine','mother','
                                                    sister','
                                                    granddaughter','
                                                    grandmother','aunt','
                                                    fiica','sora']) & df[
                                                    'derived_gender'].eq(
                                                    'male'))]
  choices = ['undetermined','undetermined']

  # where derived genders contradict relative status, set to undetermined,
                                        otherwise leave derived value
  df['derived_gender'] = np.select(conditions, choices, df['derived_gender'
                                        ])
```

```
  # where derived genders empty, fill with original genders (this wil lalso
                                        set 'undetermined' to NaN values
                                        where applicable)
  df['derived_gender'] = df['derived_gender'].fillna(df['gender'])

  # resetting type as str
  df[['gender','derived_gender']] = df[['gender','derived_gender']].astype(
                                        str)

  # removing first_firstname temporary column
  df.drop(columns=['first_firstname'], inplace=True)

  # congruency control between original genders and derived genders
  #print('\n "gender derived = gender" congruence : \n', df.groupby(['
                                        gender', 'derived_gender'])['
                                        derived_gender'].count())


  return df

def Family_Count(df):
  """Function counting the number of individuals within a household (family
                                        ) id
  :param df (dataframe): dataframe built from the source
  :returns: df with family count column
  """
  df_members = df.groupby('household', dropna='True').size().reset_index(
                                        name='members_count')
  df = df.merge(df_members, on='household', how='left')

  return df
'''
def Family_Firstnames(df):
  """Function listing the first firstname of individual within a household
                                        (family) id
  :param df (dataframe): dataframe built from the source
  :returns: df with family firstnames column
  """
  df_fam_firstnames = df.groupby('household', dropna='True')['firstname'].
                                        apply(list).reset_index(name='
                                        fam_firstnames')
  df_fam_firstnames['fam_firstnames'] = df_fam_firstnames['fam_firstnames
                                        '].apply(lambda x: sorted(str(i)
                                        for i in x))
  df_fam_firstnames['fam_firstnames'] = df_fam_firstnames['fam_firstnames
                                        '].apply(lambda x: sorted(str(i)
                                        for i in x))
```

```
  df_fam_firstnames['fam_firstnames'] = df_fam_firstnames['fam_firstnames
                                        '].apply(lambda x: ' '.join(x))
  df = df.merge(df_fam_firstnames, on='household', how='left')

  return df

def Neighbours(df):
  """Function listing the first firstname of persons immediately before and
                                         after within the registry
  :param df (dataframe): dataframe built from the source
  :returns: df with immediate neighbour firstnames column
  """
  df['neighbours'] = df['firstname'].shift(1) + ' ' + df['firstname'].shift
                                        (-1)

  return df

def Family_Firstnames_Init(df):
  """Function listing the initials (main firstnames) of individuals within
                                        a household (family) id
  :param df (dataframe): dataframe built from the source
  :returns: df with family initials column
  """
  df_fam_firstnames_init = df.groupby('household', dropna='True')['
                                        firstname'].apply(list).reset_index
                                        (name='fam_firstnames_init')
  df_fam_firstnames_init['fam_firstnames_init'] = df_fam_firstnames_init['
                                        fam_firstnames_init'].apply(lambda
                                        x: sorted(str(i)[0] for i in x))
  df = df.merge(df_fam_firstnames_init, on='household', how='left')

  return df
'''


# entirely prepare the 2 input files for the study, thanks to the above
                                      functions
def Prepare(file_a,year_a: int,file_b, year_b: int):
  """Function preparing 2 files the user would like to standardise and
                                        compare, saves them as (provided
                                        file_a).csv & (provided file_b).csv
                                        , returns them as "df_a" & "df_b"
                                        for further processes
    :param file_a (string): name without extension of the first file to be
                                          used in the study
    :param year_a (int): production year of the first file (to compute
                                          birthyear from the document's
```

```
                                        ages if any)
  :param file_b (string): name without extension of the second file to be
                                         used in the study
  :param year_a (int): production year of the second file (to compute
                                        birthyear from the document's
                                        ages if any)
  :returns: file_a (as df_a) and file_b (as df_b) cleansed and prepared
                                        for the next steps
"""
# load the files
df_a = pd.read_csv (entree_xml+'/'+ file_a + '.csv', encoding='utf8',sep=
                                        ';', index_col=False, converters={'
                                        peNumberAge-Age': Convert_Age, '
                                        peNumberAge-Residence Age':
                                        Convert_Age}, dtype = str)
df_b = pd.read_csv (entree_xml+'/'+ file_b + '.csv', encoding='utf8', sep
                                        =';', index_col=False, converters={
                                        'peNumberAge-Age': Convert_Age, '
                                        peNumberAge-Residence Age':
                                        Convert_Age}, dtype = str)

# creating the "standard_cols_dict" dictionary used by
                                        Change_Column_Names(df_a,
                                        standard_cols_dict) to standardise
                                        column labels
df_standard_cols = pd.read_excel (entree_xml+'/standard_columns2.xlsx',
                                        usecols=[0,1], dtype={'
                                        old_column_name':'object', '
                                        standard_column_name': 'object'})
standard_cols_dict = pd.Series(df_standard_cols.standard_column_name.
                                        values,index=df_standard_cols.
                                        old_column_name).to_dict()

#print('\df_a head\n',df_a.head(10))
#print('\df_b head\n',df_b.head(10))

# renaming column labels by function Change_Column_Names(df_a,
                                        standard_cols_dict) with the
                                        standard column names dictionary
df_a = Change_Column_Names(df_a,standard_cols_dict)
df_b = Change_Column_Names(df_b,standard_cols_dict)

# sorting by individuals appearing order (by id)
df_a = df_a.sort_values(by=['ushmm_id'])
df_b = df_b.sort_values(by=['ushmm_id'])

#print('\df_a head\n',df_a.head(10))
```

```
#print('\df_b head\n',df_b.head(10))

Compute_Birthyear(df_a,year_a, file_a)
Compute_Birthyear(df_b,year_b, file_b)

df_a = Normalise(df_a)
df_b = Normalise(df_b)

# expanding name abbreviations (C-tin to Constantin) from the df_spelling
                                        dictionary, built in preambule
                                        above
firstlast_cols=['firstname','lastname']
df_a.loc[:, firstlast_cols] = df_a[firstlast_cols].replace(df_spelling,
                                        regex=True)
df_b.loc[:, firstlast_cols] = df_b[firstlast_cols].replace(df_spelling,
                                        regex=True)

df_a = Appellation(df_a, firstlast_cols)
df_b = Appellation(df_b, firstlast_cols)

df_a = df_a.loc[:,~df_a.columns.str.contains('appelation', case=False)]
df_b = df_b.loc[:,~df_b.columns.str.contains('appelation', case=False)]

df_a = Remove_Irrelevant_Char_Name(df_a,firstlast_cols)
df_b = Remove_Irrelevant_Char_Name(df_b,firstlast_cols)

df_a = Dot(df_a, var = [col for col in df_a.columns if 'name' in col] )
df_b = Dot(df_b, var = [col for col in df_b.columns if 'name' in col] )


#df_a = List_Firstnames(df_a)
#df_b = List_Firstnames(df_b)

df_a = Derive_Gender(df_a)
df_b = Derive_Gender(df_b)

df_a = Split_Names(df_a,firstlast_cols)
df_b = Split_Names(df_b,firstlast_cols)
#ex dot place

df_a_cols_full = [col for col in df_a.columns if ('name' in col and not '
                                        names' in col)]
#df_a_cols_full = ['firstname','lastname']
df_a = Fullname(df_a, var = df_a_cols_full)
df_b_cols_full = [col for col in df_b.columns if ('name' in col and not '
                                        names' in col)]
#df_b_cols_full = ['firstname','lastname']
```

```
  df_b = Fullname(df_b, var = df_b_cols_full)

  df_a = Family_Count(df_a)
  df_b = Family_Count(df_b)

  #df_a = Family_Firstnames(df_a)
  #df_b = Family_Firstnames(df_b)

  #df_a = Neighbours(df_a)
  #df_b = Neighbours(df_b)

  #df_a = Family_Firstnames_Init(df_a)
  #df_b = Family_Firstnames_Init(df_b)




  # save latest study records (original data) prepared
  df_a.to_csv(entree_xml + file_a + '_prepared.csv', encoding="utf_8_sig",
                                        sep = ';', index=False)
  df_b.to_csv(entree_xml + file_b + '_prepared.csv', encoding="utf_8_sig",
                                        sep = ';', index=False)


  return df_a, df_b

def Test_Scores(df):
  """Function calculating a set of string similarities between 2 files
                                        already matched
    :param df (string): name without extension of the ids mapping (ushmm_id
                                          ) file between the 2 matched
                                          files
    :returns: df with a set of string similarities between 2 files already
                                          matched
  """
  # load the persons ids mapping file
  manual_matches = pd.read_csv (entree_xml + '/' + df + '.csv', encoding='
                                        utf8',sep=';', index_col=False,
                                        dtype = str)

  # load the files
  original_census_latest_prepared = pd.read_csv (entree_xml + '/
                                        original_census_latest_prepared.csv
                                        ', encoding='utf8',sep=';',
                                        index_col=False, dtype = str)
  original_deportation_latest_prepared = pd.read_csv (entree_xml + '/
                                        original_deportation_latest_prepared
```

```
                                        .csv', encoding='utf8',sep=';',
                                        index_col=False, dtype = str)

# merge prepared original data based on the mapping ids file
scores = pd.merge(manual_matches, original_census_latest_prepared,
                                        left_on=['census'], right_on=['
                                        ushmm_id'], how='left').drop('
                                        census', axis=1)
scores = pd.merge(scores, original_deportation_latest_prepared, left_on=[
                                        'deportation'], right_on=['ushmm_id
                                        '], how='left').drop('deportation',
                                         axis=1)

# select rows where data is available and persons not having firstname =
                                        lastname (to evaluate matches where
                                         first and last names are reversed)
names_notna = scores[scores['firstname_x'].notnull() & scores['lastname_y
                                        '].notnull() & scores['firstname_y'
                                        ].notnull() & scores['lastname_x'].
                                        notnull() & (scores['firstname_x']
                                        not in scores['lastname_x'])][['
                                        ushmm_id_x','firstname_x','
                                        lastname_x','firstname_y','
                                        lastname_y']]

# create reversed first/last names indicator based on 80% levenshstein
                                        match
names_notna['reversed'] =  names_notna[['firstname_x','lastname_x','
                                        firstname_y','lastname_y']].apply(
                                        lambda x: 'Y' if (txtlev(x[0],x[3])
                                         >= 0.8 and txtlev(x[2],x[1]) >= 0.
                                        8) else 'N', axis=1)

# only keep ids for the selection and change mergekey name (to remove it
                                        in next step)
names_notna.drop(['firstname_x','lastname_x','firstname_y','lastname_y'],
                                         axis=1, inplace=True)
names_notna.rename({'ushmm_id_x': 'temp_id'}, axis=1, inplace=True)

# append reversed indicator to the main score file, fillna with indicator
                                         N (names not reversed)
scores = pd.merge(scores, names_notna, left_on=['ushmm_id_x'], right_on=[
                                        'temp_id'], how='left').drop('
                                        temp_id', axis=1)
scores['reversed'] = scores['reversed'].fillna('N')

# get best match between individuals firstnames
```

```
scores['firstnames_fuzzy_tok_set'] = scores[['original_first_x','
                                    original_first_y']].apply(lambda x:
                                     combinations(x[0],x[1])[0:2] if(np
                                    .all(pd.notnull(x[0])) and np.all(
                                    pd.notnull(x[1]))) else None, axis=
                                    1) # if (' ' in x[0] or ' ' in x[1
                                    ]) else None
scores['best_firstname_x'] = scores['firstnames_fuzzy_tok_set'].apply(
                                    lambda x: x[0] if x else None)
scores['best_firstname_y'] = scores['firstnames_fuzzy_tok_set'].apply(
                                    lambda x: x[-1] if x else None) #
                                    isinstance(x, list)

# calculate set of scores on available data (field by field : atomic
                                    comparison)

scores['Additional_FirstNames_x'] = scores[['best_firstname_x','
                                    original_first_x']].apply(lambda x:
                                     ' '.join([str(i) for i in x[1].
                                    split() if str(i) != x[0]]) if(np.
                                    all(pd.notnull(x[0])) and np.all(pd
                                    .notnull(x[1]))) else None, axis=1)
scores['Additional_FirstNames_y'] = scores[['best_firstname_y','
                                    original_first_y']].apply(lambda x:
                                     ' '.join([str(i) for i in x[1].
                                    split() if str(i) != x[0]]) if(np.
                                    all(pd.notnull(x[0])) and np.all(pd
                                    .notnull(x[1]))) else None, axis=1)

# scores['sc_Family_Firstnames'] = scores[['fam_firstnames_x','
                                    fam_firstnames_y']].apply(lambda x:
                                     round(pyMEJW((' '.join(map(str, x[
                                    0]))).split(),(' '.join(map(str, x[
                                    1]))).split()),2) if(np.all(pd.
                                    notnull(x[0])) and np.all(pd.
                                    notnull(x[1]))) else None, axis=1)

scores['sc_best_firstname_lev'] = scores[['best_firstname_x','
                                    best_firstname_y']].apply(lambda x:
                                     round(txtlev(x[0],x[1]),2) if(np.
                                    all(pd.notnull(x[0])) and np.all(pd
                                    .notnull(x[1]))) else None, axis=1)
scores['sc_firstname_lev'] = scores[['firstname_x','firstname_y']].apply(
                                    lambda x: round(txtlev(x[0],x[1]),2
                                    ) if(np.all(pd.notnull(x[0])) and
                                    np.all(pd.notnull(x[1]))) else None
```

```
                                        , axis=1)
scores['sc_lastname_lev'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtlev(x[0],x[1]),2
                                        ) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)

scores['sc_firstname_dlev'] = scores[['firstname_x','firstname_y']].apply
                                        (lambda x: round(txtdlev(x[0],x[1])
                                        ,2) if(np.all(pd.notnull(x[0])) and
                                         np.all(pd.notnull(x[1]))) else
                                        None, axis=1)
scores['sc_lastname_dlev'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtdlev(x[0],x[1]),
                                        2) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)

scores['sc_firstname_jw'] = scores[['firstname_x','firstname_y']].apply(
                                        lambda x: round(txtjw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
scores['sc_lastname_jw'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtjw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)

scores['sc_firstname_sw'] = scores[['firstname_x','firstname_y']].apply(
                                        lambda x: round(txtsw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
scores['sc_lastname_sw'] = scores[['lastname_x','lastname_y']].apply(
                                        lambda x: round(txtsw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)

scores['sc_fullname_lev'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtlev(x[0],x[1]),2
                                        ) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)
scores['sc_fullname_dlev'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtdlev(x[0],x[1]),
```

```
                                        2) if(np.all(pd.notnull(x[0])) and
                                        np.all(pd.notnull(x[1]))) else None
                                        , axis=1)
  scores['sc_fullname_jw'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtjw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
  scores['sc_fullname_sw'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(txtsw(x[0],x[1]),2)
                                         if(np.all(pd.notnull(x[0])) and np
                                        .all(pd.notnull(x[1]))) else None,
                                        axis=1)
  # calculating Monge-Elkan only for fullname as it has the property of
                                        processing multiword comparisons
  scores['sc_fullname_ME'] = scores[['fullname_x','fullname_y']].apply(
                                        lambda x: round(pyMEJW(x[0].split()
                                        ,x[1].split()),2) if(np.all(pd.
                                        notnull(x[0])) and np.all(pd.
                                        notnull(x[1]))) else None, axis=1)

  # calculating mean of atomic scores between first and lastnames results (
                                        except Monge-Elkan as indicated)
  scores['sc_lev_avg'] = round(scores[['sc_firstname_lev', 'sc_lastname_lev
                                        ']].mean(axis=1),2)
  scores['sc_dlev_avg'] = round(scores[['sc_firstname_dlev', '
                                        sc_lastname_dlev']].mean(axis=1),2)
  scores['sc_jw_avg'] = round(scores[['sc_firstname_jw', 'sc_lastname_jw']]
                                        .mean(axis=1),2)
  scores['sc_sw_avg'] = round(scores[['sc_firstname_sw', 'sc_lastname_sw']]
                                        .mean(axis=1),2)


  # converting to float
  float_scores_cols = [col for col in scores if col.startswith('sc_')]
  scores[float_scores_cols] = scores[float_scores_cols].astype(float)

  # save latest study records (original data) prepared & scored
  scores.to_csv(entree_xml + '/scores_test.csv', encoding="utf_8_sig", sep
                                        = ';', decimal=",", index=False)

  return scores

def Record_Linkage(file_a, file_b):
  """Function performing Record Linkage Matching between (provided file_a).
                                        csv & (provided file_b).csv,
                                        returning the set of results for
```

```
                                        analysis
  :param file_a (string): name without extension of the first file to be
                                         used in Record Linkage
  :param file_b (string): name without extension of the second file to be
                                         used in Record Linkage
  :returns:
"""
# test whether the prepared files exist
try:
  df_a = pd.read_csv (entree_xml+'/'+ file_a + '.csv', encoding='utf8',
                                         sep=';', index_col=False, dtype =
                                          str)
except:
  return print('The file '+ file_a + 'does not exist, please prepare the
                                         file first and retry')
try:
  df_b = pd.read_csv (entree_xml+'/'+ file_b + '.csv', encoding='utf8',
                                         sep=';', index_col=False, dtype =
                                          str)
except:
  return print('The file '+ file_b + 'does not exist, please prepare the
                                         file first and retry')


# defining blockkeys
df_a['blockkey1'] = df_a['residence_territorial_entity']
df_b['blockkey1'] = df_b['residence_territorial_entity']

# other examples and ways of creating blocking key for the user
                                        information
# defining combinations of fields
# blockkey2_cols = ['residence_territorial_entity','birthyear','firstname
                                        ','lastname']
# creating the above fields combination
# df_a['blockkey2'] = df_a[blockkey2_cols].fillna('').sum(axis=1)

# definign a blockkey with the first 2 caracters of the territorial
                                        entity and the initial of the
                                        firstname and last caracter of
                                        firstname
# df_a['blockkey3'] = df_a['residence_territorial_entity'].str[:2] + df_a
                                        ['firstname'].str[:1] + df_a['
                                        firstname'].str[-1]

# set up indexer for Recordlinkage
indexer = recordlinkage.Index()
#indexer.full() # full index, this exceeds Colab's memory capacity
```

```
indexer.block(['blockkey1','derived_gender'],['blockkey1','derived_gender
                                        '])
#indexer.block('digital_file','digital_file')
#indexer.block('lastname','lastname')
#indexer.block('derived_gender','derived_gender') # this alone exceeds
                                        Colab's memory capacity
#indexer.block('relative','relative')
#indexer.block('birthyear','birthyear')
#indexer.add(SortedNeighbourhood(left_on='lastname', right_on='lastname',
                                         block_on=['blockkey1','
                                        derived_gender'], window=9))

compare_cols = ['ushmm_id', 'birthyear', 'firstname', 'firstname_2', '
                                        firstname_3', 'derived_gender', '
                                        lastname', 'fullname', 'relative',
                                        'household','
                                        residence_territorial_entity', '
                                        source_id', 'blockkey1']
df_a = df_a.filter(compare_cols)
df_b = df_b.filter(compare_cols)

#df_a.index = np.arange(len(df_a))
#df_b.index = np.arange(len(df_b))

df_a.set_index('ushmm_id', inplace=True, drop=True)
df_b = df_b.set_index('ushmm_id')

print('\n dfa_tomatch head\n', df_a.head(2))
print('\n dfb_tomatch head\n', df_b.head(2))

# index pairs
print('\n dfa cols :\n', df_a.columns)
print('\n dfb cols :\n', df_b.columns)

candidate_pairs = indexer.index(df_a, df_b)
print("candidate_pairs:", len(candidate_pairs))
print('\ncandidate_pairs index : \n',candidate_pairs)


# initialise class
comp = recordlinkage.Compare()

# initialise similarity measurement algorithms
comp.string('firstname', 'firstname', method='levenshtein', threshold=0.
                                        70, missing_value=0, label='
                                        firstname')
```

```
comp.string('firstname_2', 'firstname_2', method='levenshtein', threshold
                                    =0.70, missing_value=0, label='
                                    firstname2')
comp.string('lastname', 'lastname', method='levenshtein', threshold=0.80,
                                     missing_value=0, label='lastname')
comp.string('fullname', 'fullname', method='levenshtein', threshold=0.80,
                                     missing_value=0, label='fullname')
#comp.string('firstname', 'lastname', method='levenshtein', threshold=0.
                                    85, label='firstname in last')
#comp.string('lastname', 'firstname', method='levenshtein', threshold=1,
                                    label='lastname in first')
comp.exact('birthyear', 'birthyear',  missing_value=0, label='birthyear')
#comp.numeric('birthyear', 'birthyear', method='linear', offset=3, scale=
                                    3, missing_value=0.5, label='
                                    birthyear') #too much ressources
                                    needed, session crash
comp.exact('household', 'household',  missing_value=0, label='household')
comp.exact('derived_gender', 'derived_gender',  missing_value=0, label='
                                    gender')
comp.exact('relative', 'relative', missing_value=0, label='relative')
comp.exact('residence_territorial_entity', 'residence_territorial_entity'
                                    , label='
                                    residence_territorial_entity')

# the method .compute() returns the DataFrame with the feature vectors (1
                                     for fields where the rule is
                                    fullfiled, 0 otherwise).
features = comp.compute(candidate_pairs, df_a, df_b)

# displaying results statistics
# mean, standard, quantile, etc of results
print(features.describe())
# sum of pairs by number of attributes matching the rules
print(features.sum(axis=1).value_counts().sort_index(ascending=False))
# control view on screen
print(features.head(10))

# filtering potential matches where rules fullfilled total >=3 (can be
                                    adjusted) and both first &
                                    lastnames are matching the above
                                    criteria
potential_matches = features[(features.sum(axis=1) >=3) & (features[['
                                    firstname','lastname']].sum(axis=1)
                                     >1)].reset_index()
# add a column summing all results of the comparisons
potential_matches['Score'] = potential_matches.loc[:, 'firstname':'
                                    residence_territorial_entity'].sum(
```

```
                                        axis=1)
# control view on screen
print(potential_matches.head(10))

# RL toolkit only returns index and results, original data needs to be
                                        mapped back based on id
# select columns to display with the results (usually the ones used in
                                        comparison rules)
df_a_lookup = df_a[['fullname','firstname_2','birthyear','derived_gender'
                                        ,'relative','household','
                                        residence_territorial_entity','
                                        source_id']].reset_index()
df_b_lookup = df_b[['fullname','firstname_2','birthyear','derived_gender'
                                        ,'relative','household','
                                        residence_territorial_entity','
                                        source_id']].reset_index()

# adding the above columns to the results
df_a_merge = pd.merge(potential_matches, df_a_lookup, left_on='ushmm_id_1
                                        ',right_on='ushmm_id').drop('
                                        ushmm_id', axis=1)
final_merge = pd.merge(df_a_merge, df_b_lookup, left_on='ushmm_id_2',
                                        right_on='ushmm_id').drop('ushmm_id
                                        ', axis=1)
# control view on screen
print(final_merge.head(10))

# exporting results to csv
final_merge.to_csv(entree_xml+'/Record_Linkage.csv', encoding="utf_8_sig"
                                        , sep = ';', decimal=",", index=
                                        False)

#########
# Unsupervised maching and evaluation of performance
#########

feature_vectors = comp.compute(candidate_pairs, df_a, df_b)

# create training and test sets (a model should not be trained and tested
                                         on the same subset of the data)
train, test = train_test_split(feature_vectors, test_size=0.25)

# load the reference matching pairs
manual_matches = pd.read_csv (entree_xml+'/
                                        census_deportation_ground_truth.csv
                                        ', encoding='utf8',sep=';',
                                        index_col=False, dtype = str)
```

```
    manual_matches = manual_matches.set_index(['ushmm_id_x', 'ushmm_id_y']).
                                            index

    # load true pairs for the evalation of the test
    test_matches_index = test.index & manual_matches

    # built the K-mean classifier
    kmeans = recordlinkage.KMeansClassifier()

    # command for the model training
    result_kmeans = kmeans.learn(train)

    # build the predictions on the test set
    predictions = kmeans.predict(test)

    # prepare the confusion matrix (True/False positives and True/False
                                            negatives)
    confusion_matrix = recordlinkage.confusion_matrix(test_matches_index,
                                            predictions, len(test))

    # display the precision, recall and F-measure scores
    print('Précision (Precision) : ', recordlinkage.precision(
                                            confusion_matrix))
    print('Rappel (Recall) : ', recordlinkage.recall(confusion_matrix))
    print('F1-Score (F-Measure) : ', recordlinkage.fscore(confusion_matrix))

    return final_merge


#########
# Below the commands to run
# 1) the preparation and normalisation of the USHMM files
# Columns naming conventions in the source files need to follow the
                                        standard_columns.xlsx values (either
                                        old or standard)
# 2) apply if needed a set of similarity scores on the prepared files
# 3) run record linkage matching on the "latest_prepared" set of files
#########

# command 1)
Prepare('original_census_latest',1942,'original_deportation_latest',1942)

# command 2)
scores_test = Test_Scores('manual_matches')

# command 3)
```

```
Record_Linkage('original_census_latest_prepared', '
                                        original_deportation_latest_prepared'
                                        )
```

Bibliographie

Histoire de la Roumanie

Castellan (Georges), *Histoire Du Peuple Roumain*, Crozon, 2002.

Durandin (Catherine), *Histoire Des Roumains*, Paris, 1995.

Historical Regions of Romania & Neigboroughs, avec la coll. de Mariusz Paździora et Spiridon Ion Cepleanu, Creative Commons Attribution 3.0, 22 mars 2009, url : `https://commons.wikimedia.org/wiki/File:RomaniaHistRegions.jpg` (visité le 20/08/2022).

Iorga (Nicolae) et Prodan (David), *A History of Romania : Land, People, Civilization*, Havertown, UNITED STATES, 2019, url : `http://ebookcentral.proquest.com/lib/univpsl-ebooks/detail.action?docID=6465207` (visité le 21/08/2022).

Sandu (Traian), *Histoire de la Roumanie*, Paris, 2008.

Contexte historique - La déportation

About (Ilsen) et Abakunova (Anna), *The Genocide and Persecution of Roma and Sinti. Bibliography and Historiographical Review,* Research Report, International Holocaust and Remembrance Alliance, 2016, p. 139, URL : https://hal.archives-ouvertes.fr/hal-02529522 (visité le 17/08/2022).

Achim (Viorel), *The Roma in Romanian History*, Budapest, HUNGARY, 2004, URL : http://ebookcentral.proquest.com/lib/univpsl-ebooks/detail.action?docID=3137208 (visité le 17/08/2022).

— « La déportation des Rroms en Transnistrie, les données principales », *Études Tsiganes*, 56–57–1-2 (2016), p. 66-89, DOI : 10.3917/tsig.056.0066.

Anastasoaie (Marian Viorel), « Roma/Gypsies in the History of Romania : An Old Challenge for Romanian Historiography », *Romanian Journal of Society and Politics, Vol. 3, no. 1* (, 1er mai 2003), URL : https://www.academia.edu/2534874/Roma_Gypsies_in_the_History_of_Romania_An_Old_Challenge_for_Romanian_Historiography (visité le 18/08/2022).

Asséo (Henriette), « L'avènement politique des Roms (Tsiganes) et le génocide. La construction mémorielle en Allemagne et en France », *Le Temps des médias*, 5–2 (2005), p. 78-91, DOI : 10.3917/tdm.005.0078.

Beck (Sam), « The Origins of Gipsy Slavery in Romania », *Dialectical Anthropology*, 14–1 (1989), p. 53-61, JSTOR : 29790296.

Benjamin (Lya), « La politique antijuive du régime Antonescu (1940-1944) relative aux juifs de l'ancien royaume et du sud de la Transylvanie », trad. par Andreea Rota, *Revue d'Histoire de la Shoah*, 194–1 (2011), p. 27-62, DOI : 10.3917/rhsho.194.0027.

Blasen (Philippe Henri), « De La Nomination Du Cabinet Goga Au Coup d État Du Roi Carol II (28 Décembre 1937 - 10 Février 1938) », *Studia Universitatis Babeș-Bolyai Historia*, 63–2 (2018), p. 111, URL : https://www.academia.edu/38211581/De_la_nomination_du_cabinet_Goga_au_coup_d%CA%BC%C3%89tat_du_roi_Carol_II_28_d%C3%A9cembre_1937_10_f%C3%A9vrier_1938_ (visité le 23/08/2022).

Chiriac (Bogdan), « Mihail Kogălniceanu's Historical Inquiry into the Question of Roma Slavery in Mid-Nineteenth-Century Romanian Principalities », *Critical Romani Studies*, 2 (2 déc. 2020), p. 24-41, DOI : 10.29098/crs.v2i2.64.

CSOP (TNS), *Final Report : Qualitative Survey (Focus Groups) Attitudes towards the Roma in Romania*, report, World Bank, 2005, URL : https://tandis.odihr.pl/handle/20.500.12389/19969 (visité le 18/08/2022).

Égalité, inclusion et participation des Roms dans l'UE, Commission européenne - European Commission, URL : https://ec.europa.eu/info/policies/justice-and-fundamental-rights/combatting-discrimination/roma-eu/roma-equality-inclusion-and-participation-eu_fr (visité le 21/08/2022).

Fiches d'informations sur l'histoire des Roms, Roms et Gens du voyage, URL : https://www.coe.int/fr/web/roma-and-travellers/roma-history-factsheets (visité le 19/08/2022).

Fraser (Angus M.), *The Gypsies*, 2nd ed, Oxford, UK ; Cambridge, USA, 1995 (The Peoples of Europe).

Hancock (Ian F.), *The Pariah Syndrome : An Account of Gypsy Slavery and Persecution*, 2nd rev. ed., with an index, Ann Arbor, 1987.

Haynes (Rebecca), « Germany and the Establishment of the Romanian National Legionary State, September 1940 », *The Slavonic and East European Review*, 77–4 (1999), p. 700-725, JSTOR : 4212960.

International Commision on the Holocaust in Romania, Tuvia Friling, Radu Ioanid et Mihail E. Ionescu (éd.), *Final Report*, Iași, 2005.

Hübschmannová (M.), « What Can Sociology Suggest About the Origin of Roms », *Archív Orientální*, 40 (1972), p. 51-64, URL : https://www.proquest.com/docview/1304094622/citation/5F1D1CF72AE4A49PQ/1 (visité le 19/08/2022).

Kelso (Michelle), « Recognizing the Roma : A Study of the Holocaust as Viewed in Romania. » (1er janv. 2010).

— « 'And Roma Were Victims, Too.' The Romani Genocide and Holocaust Education in Romania », *Intercultural Education*, 24–1-02 (24 mai 2013), p. 61-78, DOI : 10.1080/14675986.2013.768060.

Kenrick (Donald) et Donald (Kenrick), *Gypsies, from the Ganges to the Thames*, Hatfield, Hertfordshire, 2004 (Interface Collection, 3).

Kenrick (Donald) et Taylor (Gillian), *Historical Dictionary of the Gypsies (Romanies)*, Lanham, Md, 1998 (European Historical Dictionaries, no. 27).

Kogălniceanu (Mihail (1817-1891) Auteur du texte), *Esquisse sur l'histoire, les moeurs et la langue des Cigains, connus en France sous le nom de Bohémiens : suivie d'un recueil de sept cents mots cigains / par Michel de Kogalnitchan*, 1837, URL : https://gallica.bnf.fr/ark:/12148/bpt6k34131463 (visité le 19/08/2022).

Liégeois (Jean-Pierre), *Roms et Tsiganes :* 2019 (Repères), DOI : 10.3917/dec.liege.2019.01.

MARUSHIAKOVA (Elena), « Roma / Gypsies in Ottoman Empire » (), URL : https://www.academia.edu/1132441/Roma_Gypsies_in_Ottoman_Empire (visité le 19/08/2022).

MARUSHIAKOVA (Elena) et POPOV (Vesselin), « Gypsy Slavery in Wallachia and Moldavia. » (25 févr. 2013).

— « 'Letter to Stalin' : Roma Activism vs. Gypsy Nomadism in Central, South-Eastern and Eastern Europe before WWII », *Social Inclusion*, 8–2 (4 juin 2020), p. 265-276, DOI : 10.17645/si.v8i2.2777.

MATEI (Petre), « Between Nationalism and Pragmatism : The Roma Movement in Interwar Romania », *Social Inclusion*, 8–2 (2020), p. 305, URL : https://www.academia.edu/43300578/Between_Nationalism_and_Pragmatism_The_Roma_Movement_in_Interwar_Romania (visité le 18/08/2022).

MATRAS (Yaron), *Romani : A Linguistic Introduction*, Cambridge, 2002.

NASTASĂ (Lucian) et VARGA (Andrea), *The Volume of Documents "Gypsies of Romania (1919-1944)"*, 2001, URL : https://roma-survivors.ro/en/resources/volumul-de-documente-tiganii-din-romania-1919-1944 (visité le 18/08/2022).

Romania, URL : https://encyclopedia.ushmm.org/content/en/article/romania (visité le 17/08/2022).

SANDU (Traian), « La Roumanie, une victoire à la Pyrrhus », *Les cahiers Irice*, 13–1 (2015), p. 155-170, DOI : 10.3917/lci.013.0155.

SÎRBU (Tatiana), « Gospodar ou déporté : la catégorisation comme instrument de la déportation des Tsiganes en Transnistrie », *Études Tsiganes*, 56–57–1-2 (2016), p. 90-103, DOI : 10.3917/tsig.056.0090.

SOULIS (George C.), « The Gypsies in the Byzantine Empire and the Balkans in the Late Middle Ages », *Dumbarton Oaks Papers*, 15 (1961), p. 141-165, DOI : 10.2307/1291178, JSTOR : 1291178.

The "Majority Question" in Interwar Romania : Making Majorities from Minorities in a Heterogeneous State, The Myth of Homogeneity, 22 avr. 2020, URL : https://themythofhomogeneity.org/2020/04/22/the-majority-question-in-interwar-romania-making-majorities-from-minorities-in-a-heterogeneous-state/ (visité le 23/08/2022).

THORNE (M. Benjamin), « Assimilation, Invisibility, and the Eugenic Turn in the "Gypsy Question" in Romanian Society, 1938–1942 », *Romani Studies*, 21–2 (2011), p. 177-205, URL : https://muse.jhu.edu/article/456342 (visité le 18/08/2022).

URTIZBEREA (J. Andoni), LOCHMULLER (Hanns) et TOURNEV (Ivailo), « [Myology and Ethnic Minorities : All Roads Lead to the Roma] », *Medecine sciences : M/S*, 31 Spe 3 (7 nov. 2015), p. 34-38, DOI : 10.1051/medsci/201531s310.

WEDEKIND (Michael), « THE MATHEMATIZATION OF THE HUMAN BEING : ANTHROPOLOGY AND ETHNO-POLITICS IN ROMANIA DURING THE LATE

1930s AND EARLY 1940s », *New Zealand Slavonic Journal*, 44 (2010), p. 27-67, JSTOR : 41759355.

Woodcock (Shannon), « Romanian Romani Resistance to Genocide in the Matrix of the 'Ţigan' Other », *Anthropology of East Europe Review*, 25–2 (2[2007]), p. 28-43, URL : https://scholarworks.iu.edu/journals/index.php/aeer/article/view/404 (visité le 18/08/2022).

Zaharia (Gheorghe), « La Vie Politique En Roumanie (1940-1944) », *Revue d'histoire de la Deuxième Guerre mondiale et des conflits contemporains*, 35–140 (1985), p. 53-68, JSTOR : 25729301.

Appariements (*Record Linkage*)

A Graph Matching Method for Historical Census Household Linkage | SpringerLink, URL : https://link.springer.com/chapter/10.1007/978-3-319-06608-0_40 (visité le 04/09/2022).

Ananthakrishna (Rohit), Chaudhuri (Surajit) et Ganti (Venkatesh), « Eliminating Fuzzy Duplicates in Data Warehouses » (, 2 août 2002), DOI : 10.1016/B978-155860869-6/50058-5.

Ariel (A.), Bakker (B.F.M.), Groot (M. de), Grootheest (G. van), Laan (J. van der), Smit (J.) et Verkerk (B.), *Record Linkage in Health Data : A Simulation Study*, The Hague, 2015.

Bakhtouchi (Abdelghani), « Data Reconciliation and Fusion Methods : A Survey », *Applied Computing and Informatics*, 18–3/4 (1er janv. 2020), p. 182-194, DOI : 10.1016/j.aci.2019.07.001.

Chein (Michel), Leclère (Michel) et Nicolas (Yann), « SudocAD : A Knowledge-Based System for the Author Linkage Problem », dans *Knowledge and Systems Engineering*, dir. Van Nam Huynh, Thierry Denoeux, Dang Hung Tran, Anh Cuong Le et Son Bao Pham, Cham, 2014 (Advances in Intelligent Systems and Computing), p. 65-83, DOI : 10.1007/978-3-319-02741-8_8.

Christen (Peter), « A Comparison of Personal Name Matching : Techniques and Practical Issues », dans *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, 2006, p. 290-294, DOI : 10.1109/ICDMW.2006.2.

— « A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication », *IEEE Transactions on Knowledge and Data Engineering*, 24–9 (2012), p. 1537, URL : https://www.academia.edu/8358307/A_Survey_of_Indexing_Techniques_for_Scalable_Record_Linkage_and_Deduplication (visité le 02/09/2022).

— *Data Matching : Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, Berlin ; New York, 2012.

— *Linking Sensitive Data*, 2021, URL : https://link.springer.com/book/10.1007/978-3-030-59706-1 (visité le 26/08/2022).

Christensen (Harold T.), « I. The Method of Record Linkage Applied to Family Data », *Marriage and Family Living*, 20–1 (1958), p. 38-43, DOI : 10.2307/347362, JSTOR : 347362.

Comparison of the Text Distance Metrics, ActiveWizards : data science and engineering lab, URL : https://activewizards.com/blog/comparison-of-the-text-distance-metrics/ (visité le 31/08/2022).

COQUIO (Catherine) et POUEYTO (Jean-Luc), « Les Tsiganes et l'Europe », dans *Roms, Tsiganes, Nomades*, 2014, p. 7-60, DOI : 10.3917/kart.coqu.2014.01.0007.

DUNN (Halbert L.), « Record Linkage », *American Journal of Public Health and the Nations Health*, 36–12 (déc. 1946), p. 1412-1416, pmid : 18016455, URL : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1624512/ (visité le 25/08/2022).

DUPÂQUIER (Jacques) et DUPÂQUIER (Michel), *Histoire de La Démographie*, 1985, DOI : 10.3917/perri.dupaq.1985.01.

FELLEGI (Ivan P.) et SUNTER (Alan B.), « A Theory for Record Linkage », *Journal of the American Statistical Association*, 64–328 (1969), p. 1183-1210, DOI : 10.2307/2286061, JSTOR : 2286061.

Fichier central. Archives nationales, carnet de recherche. URL : https://labarchiv.hypotheses.org/tag/fichier-central (visité le 06/09/2022).

FU (Zhichun), ZHOU (Jun), CHRISTEN (Peter) et BOOT (Mac), « M. : Multiple Instance Learning for Group Record Linkage », dans *PAKDD 2012, Part I. LNCS*, 2012, p. 171-182.

GUHA (Sudipto), KOUDAS (Nick), MARATHE (Amit) et SRIVASTAVA (Divesh), « Merging the Results of Approximate Match Operations », dans *In VLDB*, 2004, p. 636-647.

HAND (David) et CHRISTEN (Peter), « A Note on Using the F-measure for Evaluating Record Linkage Algorithms », *Statistics and Computing*, 28–3 (mai 2018), p. 539-547, DOI : 10.1007/s11222-017-9746-6.

HERNÁNDEZ (Mauricio A.) et STOLFO (Salvatore J.), « Real-World Data Is Dirty : Data Cleansing and The Merge/Purge Problem », *Data Mining and Knowledge Discovery*, 2–1 (1er janv. 1998), p. 9-37, DOI : 10.1023/A:1009761603038.

HERZOG (Thomas N.), SCHEUREN (Fritz) et WINKLER (William E.), *Data Quality and Record Linkage Techniques*, New York ; London, 2007.

JARO (Matthew A.), « Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida », *Journal of the American Statistical Association*, 84–406 (1989), p. 414-420, DOI : 10.2307/2289924, JSTOR : 2289924.

JUREK-LOUGHREY (Anna) et P (Deepak), « Semi-Supervised and Unsupervised Approaches to Record Pairs Classification in Multi-Source Data Linkage », dans 2019, p. 55-78, DOI : 10.1007/978-3-030-01872-6_3.

KIRIELLE (Nishadi), CHRISTEN (Peter) et RANBADUGE (Thilina), « TransER : Homogeneous Transfer Learning for Entity Resolution », dans *Proceedings of the 25th International Conference on Extending Database Technology 2022, Edinburgh*, URL : https://openproceedings.org/2022/conf/edbt/paper-27.pdf (visité le 26/08/2022).

Kirielle (Nishadi), Nanayakkara (Charini), Christen (Peter), Dibben (Chris), Williamson (Lee), Garrett (Eilidh) et Manson (Clair), « Unsupervised Graph-based Entity Resolution for Accurate and Efficient Family Pedigree Search », dans *Proceedings of the 25th International Conference on Extending Database Technology 2022, Edinburgh*, URL : https://openproceedings.org/2022/conf/edbt/paper-86.pdf (visité le 26/08/2022).

Knyazeva (Anna), Kolobov (Oleg) et Turchanovsky (Igor), « An Example of Empirical Approach for Bibliographic Record Linkage », dans *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, 2016, p. 1-6, DOI : 10.1109/RCIS.2016.7549290.

Kouahla (Zineddine), *Indexation dans les espaces métriques Index arborescent et parallélisation*, thèse de doct., Université de Nantes, 2013, URL : https://tel.archives-ouvertes.fr/tel-00912743 (visité le 30/08/2022).

— « Indexation dans les espaces métriques Index arborescent et parallélisation » (), p. 215.

Lehti (Patrick) et Fankhauser (Peter), « Unsupervised Duplicate Detection Using Sample Non-duplicates », *Lecture Notes in Computer Science*, 4244 (1er janv. 2006), p. 136-164, DOI : 10.1007/11890591_5.

Moreau (Erwan), Yvon (François) et Cappé (Olivier), « Robust Similarity Measures for Named Entities Matching », dans *Proceedings of the 22nd International Conference on Computational Linguistics*, Manchester, United Kingdom, 2008, t. 1, p. 593-600, DOI : 10.3115/1599081.1599156.

— « Appariement d'entités nommées coréférentes : combinaisons de mesures de similarité par apprentissage supervisé » (), p. 11.

Newcombe (H. B.), Kennedy (J. M.), Axford (S. J.) et James (A. P.), « Automatic Linkage of Vital Records », *Science*, 130–3381 (1959), p. 954-959, JSTOR : 1756667.

Newcombe (Howard B.), « Couplage de données pour les études démographiques », *Population*, 24–4 (1969), p. 653-684, DOI : 10.2307/1527541.

Nouvelles Procédures Pour Les Appariements de Données – Le CASD – Centre d'accès Sécurisé Aux Données, URL : https://www.casd.eu/nouvelles-procedures-pour-les-appariements-de-donnees/ (visité le 05/09/2022).

Objectifs | QUALINCA, URL : https://www.lirmm.fr/qualinca/index8b01.html?q=fr/objectifs (visité le 06/09/2022).

On (Byung-Won), « Social Network Analysis on Name Disambiguation and More », dans 2008, p. 1081-1088, DOI : 10.1109/ICCIT.2008.210.

On (Byung-Won), Koudas (Nick), Lee (Dongwon) et Srivastava (Divesh), « Group Linkage », dans *2007 IEEE 23rd International Conference on Data Engineering*, Istanbul, 2007, p. 496-505, DOI : 10.1109/ICDE.2007.367895.

Papadakis (George), Skoutas (Dimitrios), Thanos (Emmanouil) et Palpanas (Themis), « Blocking and Filtering Techniques for Entity Resolution : A Survey », *ACM Computing Surveys*, 53–2 (31 mars 2021), p. 1-42, doi : 10.1145/3377455.

Picard (Nathalie) et Gadouche (Kamel), *L'accès Aux Données Très Détaillées Pour La Recherche Scientifique*, 2017-06, THEMA (THéorie Economique, Modélisation et Applications), Université de Cergy-Pontoise, 2017, url : https://ideas.repec.org/p/ema/worpap/2017-06.html (visité le 06/09/2022).

Pita (Robespierre), Sena (Samila), Fiaccone (Rosemeire), Amorim (Leila), Barreto (Mauricio), Denaxas (Spiros) et Barreto (Marcos), « Applying Machine Learning to Improve the Accuracy of Probabilistic Linkage », dans 2017.

POPP. Projet d'océrisation des recensements parisiens, url : https://popp.hypotheses.org/ (visité le 06/09/2022).

Qualité et interopérabilité de grands catalogues documentaires, Agence nationale de la recherche, url : https://anr.fr/Projet-ANR-12-CORD-0012 (visité le 05/09/2022).

Ramadan (Banda), Christen (Peter), Liang (Huizhi) et Gayler (Ross), « Dynamic Sorted Neighborhood Indexing Technique for Real-Time Entity Resolution », *The ACM Journal of Data and Information Quality (JDIQ)*, 6 (1er oct. 2015), doi : 10.1145/2816821.

— « Dynamic Sorted Neighborhood Indexing Technique for Real-Time Entity Resolution », *The ACM Journal of Data and Information Quality (JDIQ)*, 6 (1er oct. 2015), doi : 10.1145/2816821.

Romania, 1942, United States Holocaust Memorial Museum Archives, Washington, DC., url : https://encyclopedia.ushmm.org/content/en/map/romania-1942 (visité le 19/08/2022).

Ruggles (Steven), Fitch (Catherine) et Roberts (Evan), « Historical Census Record Linkage », *Annual review of sociology*, 44 (juill. 2018), p. 19-37, doi : 10.1146/annurev-soc-073117-041447, pmid : 30369709.

Sadinle (Mauricio) et Fienberg (Stephen E.), « A Generalized Fellegi–Sunter Framework for Multiple Record Linkage With Application to Homicide Record Systems », *Journal of the American Statistical Association*, 108–502 (2013), p. 385-397, JSTOR : 24246450.

Sayers (Adrian), Ben-Shlomo (Yoav), Blom (Ashley W) et Steele (Fiona), « Probabilistic Record Linkage », *International Journal of Epidemiology*, 45–3 (1er juin 2016), p. 954-964, doi : 10.1093/ije/dyv322.

Short (Jack) et Caulfield (Brian), « Record Linkage for Road Traffic Injuries in Ireland Using Police Hospital and Injury Claims Data », *Journal of Safety Research*, 58 (1er sept. 2016), p. 1-14, doi : 10.1016/j.jsr.2016.05.002.

Smith (Merran) et Flack (Felicity), « Data Linkage in Australia : The First 50 Years », *International Journal of Environmental Research and Public Health*, 18–21 (28 oct. 2021), p. 11339, doi : 10.3390/ijerph182111339, pmid : 34769852.

Tai (Xiao Hui), « Record Linkage and Matching Problems in Forensics », dans *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2018, p. 510-517, doi : 10.1109/ICDMW.2018.00081.

Un Outil d'appariement Sur Identifiants Indirects – Courrier Des Statistiques N6 - 2021 | Insee, url : https://www.insee.fr/fr/information/5398689?sommaire=5398695 (visité le 05/09/2022).

Wang (Jiannan), Li (Guoliang), Yu (Jeffrey Xu) et Feng (Jianhua), « Entity Matching : How Similar Is Similar », *Proceedings of the VLDB Endowment*, 4–10 (1er juill. 2011), p. 622-633, doi : 10.14778/2021017.2021020.

Winkler (William) et Gov (William), « Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage », *Journal of the American Statistical Association* (, 18 janv. 2002).

Références techniques

Algorithme espérance-maximisation, Wikipédia, 1er mars 2022, URL : https://fr.wikipedia.org/w/index.php?title=Algorithme_esp%C3%A9rance-maximisation&oldid=191518930 (visité le 05/09/2022).

Algorithme hongrois, Wikipédia, 8 juill. 2022, URL : https://fr.wikipedia.org/w/index.php?title=Algorithme_hongrois&oldid=195166805 (visité le 05/09/2022).

Apprentissage automatique, Wikipédia, 2 sept. 2022, URL : https://fr.wikipedia.org/w/index.php?title=Apprentissage_automatique&oldid=196639745 (visité le 05/09/2022).

Apprentissage non supervisé, Wikipédia, 1er mai 2022, URL : https://fr.wikipedia.org/w/index.php?title=Apprentissage_non_supervis%C3%A9&oldid=193330757 (visité le 05/09/2022).

Apprentissage par renforcement, Wikipédia, 21 juin 2022, URL : https://fr.wikipedia.org/w/index.php?title=Apprentissage_par_renforcement&oldid=194726951 (visité le 05/09/2022).

Apprentissage supervisé, Wikipédia, 17 mai 2022, URL : https://fr.wikipedia.org/w/index.php?title=Apprentissage_supervis%C3%A9&oldid=193754326 (visité le 05/09/2022).

Data-Matching-Software : A List of Free Data Matching and Record Linkage Software. URL : https://github.com/J535D165/data-matching-software (visité le 06/09/2022).

De Bruin (J), *Python Record Linkage Toolkit : A Toolkit for Record Linkage and Duplicate Detection in Python*, version v0.14, Zenodo, 1er déc. 2019, DOI : 10.5281/ZENODO.3559043.

Distance de Damerau-Levenshtein, Wikipédia, 17 août 2020, URL : https://fr.wikipedia.org/w/index.php?title=Distance_de_Damerau-Levenshtein&oldid=173905600 (visité le 31/08/2022).

Distance de Jaro-Winkler, Wikipédia, 22 oct. 2021, URL : https://fr.wikipedia.org/w/index.php?title=Distance_de_Jaro-Winkler&oldid=187349874 (visité le 31/08/2022).

Distance de Levenshtein, Wikipédia, 13 août 2022, URL : https://fr.wikipedia.org/w/index.php?title=Distance_de_Levenshtein&oldid=196094248 (visité le 31/08/2022).

Indice et distance de Jaccard, Wikipédia, 28 mai 2021, URL : https://fr.wikipedia.org/w/index.php?title=Indice_et_distance_de_Jaccard&oldid=183350601 (visité le 31/08/2022).

K-moyennes, Wikipédia, 1er mai 2022, URL : https://fr.wikipedia.org/w/index.php?title=K-moyennes&oldid=193331134 (visité le 05/09/2022).

Machine à vecteurs de support, Wikipédia, 24 mars 2022, URL : https://fr.wikipedia.org/w/index.php?title=Machine_%C3%A0_vecteurs_de_support&oldid=192207500 (visité le 05/09/2022).

N-*Gram*, Wikipedia, 28 août 2022, URL : https://en.wikipedia.org/w/index.php?title=N-gram&oldid=1107215398 (visité le 03/09/2022).

Partitionnement de données, Wikipédia, 5 août 2022, URL : https://fr.wikipedia.org/w/index.php?title=Partitionnement_de_donn%C3%A9es&oldid=195865450 (visité le 05/09/2022).

Plus longue sous-chaîne commune, Wikipédia, 31 oct. 2021, URL : https://fr.wikipedia.org/w/index.php?title=Plus_longue_sous-cha%C3%AEne_commune&oldid=187600673 (visité le 04/09/2022).

Précision et rappel, Wikipédia, 30 mai 2022, URL : https://fr.wikipedia.org/w/index.php?title=Pr%C3%A9cision_et_rappel&oldid=194125713 (visité le 05/09/2022).

Random Forest, Wikipedia, 30 août 2022, URL : https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=1107566819 (visité le 05/09/2022).

Regroupement hiérarchique, Wikipédia, 5 août 2022, URL : https://fr.wikipedia.org/w/index.php?title=Regroupement_hi%C3%A9rarchique&oldid=195865980 (visité le 05/09/2022).

ROPELADDER, *Record Linkage Resources*, 24 août 2022, URL : https://github.com/ropeladder/record-linkage-resources (visité le 06/09/2022).

Sensitivity and Specificity, Wikipedia, 22 août 2022, URL : https://en.wikipedia.org/w/index.php?title=Sensitivity_and_specificity&oldid=1105990559 (visité le 05/09/2022).

Smith–Waterman Algorithm, Wikipedia, 10 juill. 2022, URL : https://en.wikipedia.org/w/index.php?title=Smith%E2%80%93Waterman_algorithm&oldid=1097451242 (visité le 31/08/2022).

Soundex, FamilySearch Wiki, 24 juin 2022, URL : https://www.familysearch.org/en/wiki/Soundex (visité le 30/08/2022).

TF-IDF, Wikipédia, 5 janv. 2022, URL : https://fr.wikipedia.org/w/index.php?title=TF-IDF&oldid=189586794 (visité le 31/08/2022).

Les sources de l'USHMM

RG-25.050M, *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum Archives, Washington, DC., URL : https://collections.ushmm.org/search/catalog/irn36431 (visité le 16/08/2022).

RG-25.050M (File ID : 45835), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : https://www.ushmm.org/online/hsv/source_view.php?SourceId=45835 (visité le 10/09/2022).

RG-25.050M (File ID : 45869), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : https://www.ushmm.org/online/hsv/source_view.php?SourceId=45869 (visité le 10/09/2022).

RG-25.050M (File ID : 45931), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : https://www.ushmm.org/online/hsv/source_view.php?SourceId=45931 (visité le 07/09/2022).

RG-25.050M (File ID : 45964), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : https://www.ushmm.org/online/hsv/source_view.php?SourceId=45964 (visité le 08/09/2022).

RG-25.050M (File ID : 46036), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : https://www.ushmm.org/online/hsv/source_view.php?SourceId=46036 (visité le 08/09/2022).

RG-25.050M (File ID : 46415), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : https://www.ushmm.org/online/hsv/source_view.php?SourceId=46415 (visité le 08/09/2022).

RG-25.050M (File ID : 46521), *Selected Records from Various Archives of Romania Concerning Roma*, United States Holocaust Memorial Museum, URL : https://www.ushmm.org/online/hsv/source_view.php?SourceId=46521 (visité le 08/09/2022).