Séance 1 : créer un script Python de couplage d'enregistrements

Bonjour 👋 !

Bienvenue dans la première partie de la séquence dédiée au développement d'une **interface en ligne de commande** de **couplage d'enregistrements** de fichiers de données textuelles organisées en tables.

Objectifs de la séance 🎯

- découvrir le couplage d'enregistrements pas à pas ;
- se familiariser avec la manipulation de chaînes de caractères en Python ;
- découvrir & expérimenter une métrique usuelle de comparaison de chaînes de caractères : la distance de Levenstein ;
- créer et implémenter un algorithme de couplage d'enregistrements ;
- adapter l'algorithme à des données "réelles" imparfaites produites par un véritable projet d'humanités numériques.

> **Important**
>
> 1. Répondre aux questions de code en comlétant le fichier de fichier de script Python `couplage.py`.
> 2. 🆘 Une question n'est pas claire ? Vous êtes bloqué(e) ? N'attendez pas, **appelez à l'aide** 🙋. Le fichier `couplage.py` contient aussi des astuces et aides complémentaires.
> 3. 🤖 Vous pouvez utiliser ChatGPT/Gemini/etc. pour vous aider, **mais** contraignez vous à n'utiliser ses propositions **que si vous les comprenez vraiment**. Ne devenez pas esclave de la machine ! 🙏
> 4. 😱 Certaines sections contiennent des formules mathématiques mais **pas de panique** : (1) elles sont plus faciles à implémenter qu'elles n'en ont l'air et (2) vous serez aidé(e) pas à pas. Si quelque chose n'est pas clair, voir les points 2 & 3.
> 5. 😌 Si vous n'avez pas réussi ou pas eu le temps de répondre à une question, **pas de panique**, le répertoire `correction/` contient une solution !

> **Tip**
>
> La difficulté d'une question ✏️ est indiquée de ⭐ à ⭐⭐⭐⭐.

A/ Introduction au couplage d'enregistrements (Record Linkage)

Dans un monde parfait… 🌈🦄

Lorsque l'on travaille avec des données numériques, il arrive souvent que l'on ait besoin de **combiner** deux ensembles de données qui contiennent des informations sur les mêmes entités.
Dans un monde parfait, les entités possèdent un **identifiant** unique, stable et non ambigu. Pour combiner les deux tables il suffit donc de faire correspondre les enregistrements qui possèdent le même **identifiant**.
Dans le vocabulaire des bases de données, on apelle cette opération une **jointure**.

Voici par exemple deux extraits de catalogues stellaires contenant les **enregistrements** des étoiles les plus proches du soleil :

Gliese id	Nom	Distance_al	Type
GJ 559 A	Alpha Centauri A	4.37	Naine jaune
GJ 699	Étoile de Barnard	5.963	Naine rouge

Gliese id	Masse _M	Âge_Ga	Rayon_R
GJ 406	0.09-0.13	10	0.16-0.19
GJ 699	0.162	10	0.187

Comme deux catalogues indexent les étoiles avec leur identifiant unique Gliese, les combiner est trivial : il suffit de coupler les enregistrements avec le même `Gliese_id`.
Facile, non ? 😎

La dure réalité 😢

Évidemment, ce n'est souvent pas aussi simple.
Les problèmes commencent lorsque les enregistrements n'ont pas d'identifiant unique clair, que les informations sont incomplètes, bruitées, erronées, bref le genre de situation typique des données en humanités numériques, produites par des processus de traitement ou extraction plus ou moins automatiques !

Voici deux extraits de données issues de sources historiques imprimées, les éditions de 1855 et 1856 de "l'Annuaire général du commerce et de l'industrie de la ville de Bordeaux [...]". C'est, pour simplifier, l'équivalent des "pages jaunes" actuelles.

Table A : extraits des "Fournisseurs de Chapellerie", 1855

Nom	Adresse
Duchêne et Ce	r. de la Bourse, 15
Marcon (S.) et Ce	rue du Parlement-Ste-Catherine, 30
Teindas (H.)	r. St-Remi, 32
Vallet (V.)	pl. du Parlement, 8

Table B : extraits des "Fournisseurs de Chapellerie", 1856

Nom	Adresse
Charrier (Ve)	r. Notre-Dame, 74
Marcon (S.) et Ce	rue du Parlement-Ste-Catherine, 30
Teindas (H.)	r. St-Remi, 32
Vallet (V.)	r. du Portail, 12

> Important
>
> ✏️ **- QUESTION 1 -** ⭐
>
> Prenez une minute pour réfléchir à une stratégie permettant de déterminer si un enregistrement de la table A a un correspondant dans la table B, et discutons-en tous ensemble !

Vous venez de concevoir un premier algorithme de couplage d'enregistrements ! 👋

Posons les bases

Le *record linkage* ou *data matching* , en français "couplage d'enregistrements", désigne un ensemble de techniques servant à reconnaître, dans deux bases de données, les **enregistrements** qui correspondent à la même **entité**, qu'il s'agisse de personnes, d'objets, d'événements, etc.
Formalisé dans le champ des technologies de l'information, le couplage d'enregistrement est un classique du traitement des données en sciences sociales.
Un cas typique en histoire consiste à relier des ensembles de données extraites dans des sources d'archives pour identifier, par exemple, les références aux mêmes entités du monde réel, qu'il s'agisse de personnes, de lieux, etc.

Par souci de simplicité, on considère uniquement la situation où les deux sources de données à combiner sont des **tables**.

Une table est composées de lignes appelées **enregistrements**, et de colonnes appellés **champs**.

Pour coupler les enregistrements de deux tables, on décide d'un sous-ensemble de champs communs aux deux sources qui, ensemble, forment **l'identifiant unique** de chaque enregistrement. Dans l'exemple précédent, les **champs identifiants** `Nom` et `Adresse` forment ensemble l'identifiant unique des commerces. : si un enregistrement de A et un enregistrement de B ont les mêmes valeurs de `Nom` et `Adresse` alors ils sont considérés comme correspondants.

Le principe du couplage consiste donc à créer des liens entre enregistrements issus de deux tables et de tester si leurs identifiants correspondent. Lorsque c'est le cas on dit qu'il y a **accord** des deux enregistrements, ou plus simplement qu'il y a **match** (et **non-match** dans le cas inverse).

Le couplage d'enregistrement est un **processus**, automatisable à l'aide d'un algorithme. Il en existe un grand nombre, mais ils se rangent dans seulement deux catégories :

- le couplage **déterministe** où l'on applique des règles de correspondance binaires (oui/non) sur les **champs identifiants** de deux enregistrements pour prendre une décision binaire "tout ou rien" ;
- le couplage **probabiliste** qui utilise un modèle mathématique pour associer une probabilité de *match* entre deux enregistrements, puis prendre une décision finale en considérant les *matches* probables.

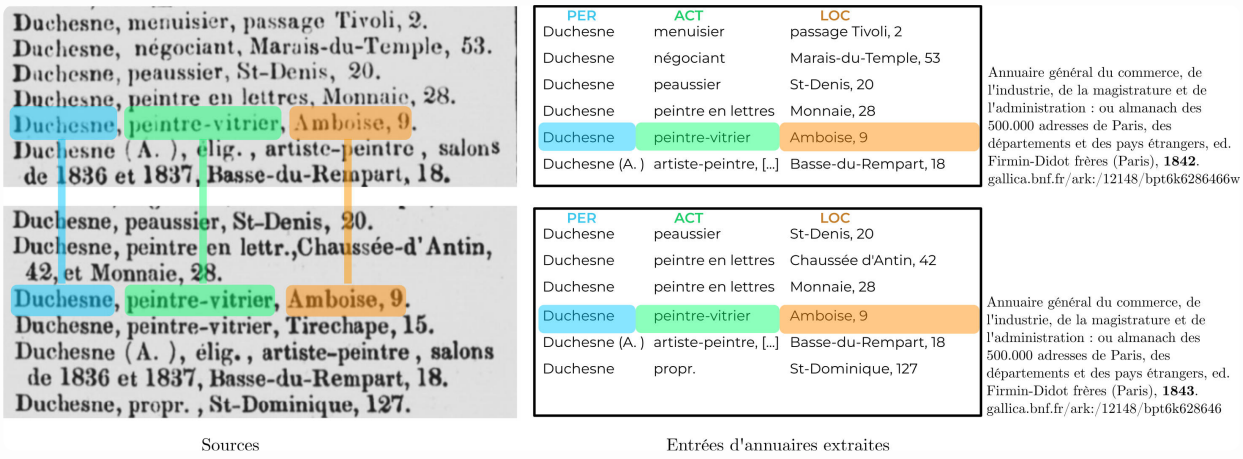
Les méthodes déterministes sont simples, mais elles ont tendance à créer de nombreux faux négatifs (on rate des couplages) lorsque les données sont imparfaites, bruitées ou incomplètes.
Les méthodes probabilistes tentent de surmonter ce problème en associant un nombre réel aux possibles couplages, permettant une prise décision plus fine et moins sensible aux imperfections des données.
Aujourd'hui, cette tâche est également souvent réalisée avec des méthodes par apprentissage, "classique" (comme SVM), voire par réseaux de neurones profonds.

Voilà pour les grandes lignes, place à l'expérimentation pour mieux comprendre ! 👩‍💻

B/ Couplage déterministe exact

À partir de la fin du XVIIIe siècle et jusqu'au XXe siècle les **annuaires du commerce** fleurissent à Paris . Ces ouvrages compilent, entre autres, les commerçants, notables et institutions urbaines dans de grandes listes imprimées souvent annuellement. Ce sont, en quelques sortes, les "pages jaunes" de l'époque.

Voici deux extraits d'annuaires de Paris, publiés par l'imprimeur Didot-Bottin en 1842 et 1843.



À l'occasion du projet de recherche SODUCO, le contenu de ces annuaires a été extrait et transformé en bases de données numériques sérielles.
Les données sont organisées en grands tableaux, un par annuaire, où chaque entrée de l'annuaire est un enregistrement contenant 3 champs :

- le **nom** (PER) de la personne/institution/commerce ;
- son **activité**, commerciale ou non (ACT) ;
- son **adresse** dans la ville, généralement au numéro (LOC).

On peut déjà faire beaucoup avec de telles données, mais on a évidemment très envie de pouvoir suivre un commerce ou une catégorie professionnelle ou au fil du temps, à travers les annuaires traités.

Voici un cas typique de couplage : il faut identifier les occurrences multiples du même commerce dans plusieurs bases de données !

Implémentation naïve

Dans l'exemple précédent, on comprend immédiatement que M. Duchesne est actif les deux années. Il est évident qu'il s'agit de la même personne car les champs PER, ACT et LOC sont identiques.

Ce couplage **exact** est le plus simple : il y a *match* si les champs identifiants (ici PER, ACT et LOC) sont exactement égaux.

Édition	PER	ACT	LOC
1842	Duchesne	peintre-vitrier	Amboise, 9

Édition	PER	ACT	LOC

1843	Duchesne	peintre-vitrier	Amboise, 9

> Important
>
> ✏️ **- QUESTION 2 - ⭐⭐**
>
> Implémentez en Python cette première méthode de couplage exact entre deux enregistrements, représentés par des listes de chaînes de caractères.
> Complétez la fonction `calculer_score_exact` qui doit comparer les champs des enregistrements deux à deux et retourner un nombre entier :
>
> - 1 si tous les champs sont deux à deux égaux ;
> - 0 si au moins une paire de champs ne sont pas égaux.

> Tip
>
> 🧪 **Sortie attendue**
>
> ```
> ✅ MATCH  :: ('Duchesne', 'Duchesne') • ('peintre-vitrier', 'peintre-vitrier')
> ```

> Note
>
> 📝 **À retenir.**
> Coupler deux enregistrements qui représentent la même entité du monde réel dans des sources de données différentes revient à vérifier que les champs qui permettent de l'identifier de manière **unique** sont les mêmes. Ici, on a considéré que deux enregistrements d'annuaires concernent la même personne si ils ont le même nom (PER), la même activité (ACT) et la même adresse (LOC).

Normaliser pour mieux comparer

En réalité, les données sont rarement immédiatement comparables. C'est d'autant plus vrai lorsqu'elles sont produites par extraction automatique, comme l'OCR, où les erreurs de reconnaissance peuvent être nombreuses. Cela peut être dû à la qualité des numérisations, aux graphies utilisées, à l'état de conservation des documents, etc.

L'exemple précédent était corrigé à la main. Voici en réalité ce qui a été extrait par OCR des annuaires.

Édition	PER	ACT	LOC
1842	duchesne,	peintre vÎtrier	amboise 9

Édition	PER	ACT	LOC

1843	DUchësne	Peintre -Vitrier	Amboise, 9.

En l'état, il est clair que la fonction `calculer_score_exact` ne va pas identifier de *match* entre ces deux enregistrements.

On a donc souvent besoin d'appliquer des **pré-traitements** pour **normaliser** les valeurs et qu'elles puissent être comparées.

> Important
>
> ✏️ **- QUESTION 3 - ⭐⭐**
>
> Quelles transformations doit on appliquer aux chaînes de caractères pour les **normaliser** et qu'elles soient de nouveau comparables avec `calculer_score_exact` ? Implémentez ces transformations en complétant la fonction `normaliser_champ`, puis testez-la avec les enregistrements ci-dessous.

> Tip
>
> 🧪 **Sortie attendue**
>
> ```
> ✅ MATCH  :: ('duchesne,,', 'DUchësne') • ('peintre vÎtrier', 'Peintre -Vitrie
> ```

> Note
>
> 📝 **À retenir.**
> Les données réelles sont rarement exemptes d'erreurs. C'est encore plus le cas de textes extraits automatiquement par OCR. Or, cela gêne fortement le couplage. Il est donc intéressant de faire précéder le couplage lui-même par des prétraitements pour "nettoyer" les enregistrements à coupler.

C/ Couplage déterministe "approximatif"

En réalité, les imperfections et erreurs qui affectent les données peuvent être bien plus grandes. Pour nos annuaires, cela peut venir entre autres :

- d'erreurs de l'OCR : lettres mal reconnues, doublées ou manquantes ;
- des véritables différences de graphie, typiques des documents historiques (ex. `Martyrs` et `Martirs`), des abréviations, etc.

Voici des extraits particulièrement bruités d'enregistrements trouvés dans les annuaires :

Édition	PER	ACT	LOC

1842	lacrox (paul, (bibliophile iaco	membre du coniite des chartes	martirs 4l

Édition	PER	ACT	LOC
1843	Lacroix Paul. (Bibliophile jacob	membre du com. des chartes	Martyrs, 47

Il est clair que normaliser ne suffira pas : il y a plusieurs erreurs de reconnaissance OCR, des caractères manquent, etc.

On pourrait adapter la fonction `normaliser_champ`, mais cela signifie qu'il faudrait prévoir en avance tous les cas d'erreurs...

Un manière de surmonter ce problème consiste à ne plus baser le couplage sur l'égalité stricte de deux enregistrements, mais à plutôt mesurer un **score de similarité** entre eux.
Ce score est un nombre réel, par exemple entre 0.0 et 1.0. Un score de 0.0 signifie que les deux enregistrements sont très différents, et de 1.0 qu'ils sont très similaires (typiquement, égaux).

L'idée est donc de calculer un score de similarité pour chaque champ et de les agréger pour obtenir un score de similarité global entre deux enregistrements.
Enfin, on décide s'il y a *match* ou non en fixant un **seuil** : une similarité supérieure à ce seuil est un *match*.

Très bien mais...comment le mesure-t-on, ce score de similarité ?! 🤔

Distance d'édition

Est-ce-que `lacrox` est plus ou moins similaire à `Lacroix` que `Lacroi` ? Quel score attribuer à ces comparaisons ?

Une manière simple de s'y prendre consiste à mesurer une **distance d'édition** entre deux chaînes de caractères, c'est à dire le **nombre minimal de modifications** à appliquer pour transformer une chaîne en l'autre. Ainsi, plus la distance d'édition est petite, plus les chaînes se "ressemblent" !

Considérons les règles de modication suivantes :

1. Ajouter un caractère, ex. `Llcarox` -> `Llcaroix`.
2. Supprimer un caractère, ex. `Llcaroix` -> `Lcaroix`.
3. Substituer deux caractères, ex. `Lcaroix` -> `Lacroix`

La distance d'édtition entre `Llcarox` et `Lacroix` et de 3, puisqu'icil faut ajouter le caractère `i`, supprimer `l` et inverser `c` et `a` dans `Llcarox` pour obtenir `Lacroix`.

> Important

> ✏️ **- QUESTION 4 -** ⭐
>
> Quelle est la distance d'édition entre les chaînes de caractères `martirs 4I` et `Martyrs, 47` ? Vérifiez à l'aide de l'outil en ligne https://fr.planetcalc.com/1721 !

Vous connaissez peut-être cette distance d'édition sous le nom de **distance de Levenshtein**, extrêmement utilisée et implémentée dans de nombreuses bibliothèques et *frameworks* logiciels.

Plusieurs bibliothèques Python implémentent la distance de Levenshtein. Nous allons utiliser NLTK, une boîte à outil dédiée au traitement automatique du langage naturel. Dans NLTK, la distance de Levenshtein `nltk.edit_distance` est disponibles dans le module `ntlk.metrics`.

> Important
>
> ✏️ **- QUESTION 5 -**
>
> Installez NLTK dans votre environnement Python courant avec `pip install` .
> Dans une console Python, importez la méthode `edit_distance` du module `nltk.metrics` et vérifiez que la distance entre les chaînes `martirs 4I` et `Martyrs, 47` soit celle attendue.

> Tip
>
> 🧪 **Sortie attendue**
>
> ```
> nltk.edit_distance('martirs 4I, 'Martyrs, 47') = 4
> ```

Similarité de chaînes de caractères

La distance d'édition est une mesure absolue des différences entre deux chaînes, elle ne mesure pas directement un **degré** de différence.
Autrement dit, une distance de 3 peut représenter des écarts très différents selon la longueur des chaînes.
Ainsi, `edit_distance('martil', 'marcel') == edit_distance('cloître St-Marcel', 'cloître Saint-Marcel' ) == 3`, pourtant cela représente dans un cas une différence de 50% et dans l'autre de seulement 15 à 17% !

Pour rendre des distances d'éditions comparables il faut les **normaliser**, c'est-à-dire les ramener dans un intervalle de valeurs fixé, par exemple dans $[0, 1]$.

Une méthode intéressante consiste à normaliser la distance par la **longueur de la plus grande des deux chaînes** :

```
d_{norm}(a,b) = \frac{edit\_distance(a,b)}{max(|a|,|b|)}
```

> Important
>
> ✏️ **- QUESTION 6 - ⭐⭐⭐**
>
> Implémentez la fonction `edit_distance_norm(str, str) -> float` qui renvoie la distance d'édition normalisée entre deux chaînes de caractères. Testez avec l'exemple des questions 5/6. Comment s'interprête le nombre obtenu ?

> Tip
>
> 🧪 **Sortie attendue**
>
> ```
> edit_distance_norm('martirs 4I, 'Martyrs, 47') = 0.36
> ```

> Tip
>
> Si a et b toutes deux vides, on aura 0 au dénominateur ($max(|a|, |b|) = 0$), ce qui va provoquer une erreur `ZeroDivisionError`. En Python la fonction `max()` accepte n paramètres, ce qui permet de se prémunir de l'erreur en passant en 3e paramètre la constante `1`. Ainsi le dénominateur sera toujours supérieur ou égal à 1, sans provoquer d'effet de bord puisque cela impacte le dénominateur uniquement lorsque a et b sont vides et donc $d_{norm}(a, b) = \frac{0}{1} = 0$.

Enfin, on aimerait transformer cette distance normalisée en **similarité**, afin qu'elle soit facilement interprétable, c'est-à-dire que $sim(a, b) = 1.0$ si $a = b$ et $sim(a, b) = 0.0$ si les chaînes a et b n'ont aucun caractère commun.

> Important
>
> ✏️ **- QUESTION 7 - ⭐**
>
> Implémentez la fonction `similarité_str(str, str) -> float` qui appelle `edit_distance_norm()` et renvoie le score de similarité correspondant à la distance calculée entre deux chaînes de caractères.

> Tip
>
> 🧪 **Sortie attendue**

> `similarité_str('martirs 4I, 'Martyrs, 47') = 0.64`

Similarité entre enregistrements

On sait maintenant calculer un score de similarité entre deux chaînes de caractères.

Un enregistrement étant composé de plusieurs champs, nous avons besoin d'une méthode pour calculer un score de similarité **entre enregistrements**, qui agrège les scores des champs qui le composent.

Une solution consiste à calculer la moyenne des scores de similarité entre champs.
Nous pourrions utiliser la moyenne arithmétique, mais elle a l'inconvénient de peu pénaliser des champs très différents si d'autres champs sont proches.

Voici un exemple des simularités entre deux enregistrements :

	PER	**ACT**	**LOC**
e1	Duchesne	peintre-vitrier	Amboise, 9
e2	Morin	peintre	Amboise, 9
Similarité	**0.13**	**0.47**	**1.00**

La moyenne arithmétique des similarités des champs PER, ACT et LOC est $\frac{0.13+0.47+1.00}{3} = 0.53$

Or on est plutôt certains qu'il ne s'agit pas du même commerce car le nom est très différent, même s'ils sont à la même adresse.
On préférerait **pénaliser** plus fortement les cas où un champ est très différent, même si les autres sont similaires.

Pour cela on peut lui préférer la **moyenne géométrique** des n champs de deux

enregistrements (ici on note sim_{c_1} la similarité calculée pour le premier champ, etc.) :

```
\overline{sim} = \sqrt[n]{sim_{c_1} \times sim_{c_2} \times ... \times sim_{c_n}}
```

Dans l'exemple précédent, on a alors :

```
\begin{equation*}
\begin{split}
\overline{sim}(e_1, e_2) & = \sqrt[3]{sim_{PER} \times sim_{ACT} \times sim_{LOC}}
& = \sqrt[3]{0.13 \times 0.47 \times 1.0}  \\
&  =  0.39
```

```
\end{split}
\end{equation*}
```

On voit que le faible score de similarité sur le champ PER a nettement pénalisé la similarité agrégée.

> Important
>
> ✏️ **- QUESTION 8 - ⭐⭐⭐**
>
> Implémentez la fonction `calculer_score_approximatif(list[str], list[str]) -> float` qui calcule et renvoie la **moyenne géométrique** des similarités entre les champs de deux enregistrements. Reportez-vous aux commentaires dans `sequence_1.py` qui vous guideront pour coder la formule en Python.

> Tip
>
> 🧪 **Sortie attendue**
>
> ```
> Similarité entre les enregistrements lacroix_1841 et lacroix_1844 : 0.73
> ```

> 📝 Bonus: pour aller plus loin.
>
> On peut rendre paramétrable la pénalisation des faibles similarités en appliquant une moyenne géométrique **pondérée** :
> $\overline{sim} = \sqrt[n]{sim_{c_1}^{\alpha} \times sim_{c_2}^{\alpha} \times \ldots \times sim_{c_n}^{\alpha}}$. Le paramètre α est un facteur de pénalisation : plus il est grand ($\alpha > 1$), plus les valeurs proches de 0 pénalisent la valeur moyenne.

Couplage approximatif

Vous voilà maintenant capables de juger plus finement de la similarité entre deux enregistrements plutôt qu'avec un indicateur binaire, trop limité.

Reste tout de même qu'in fine il faut **décider** si, entre deux enregistrements, il y a *match*, ou non. On doit donc passer d'un score de similarité mesuré par un réel entre 0 et 1, à une réponse binaire *match* / *non match*.

Une manière simple de s'y prendre consiste à fixer un **seuil de couplage** : si la similarité est supérieure à ce seuil, alors il y a *match* !

> Important
>
> ✏️ **- QUESTION 9 - ⭐⭐⭐**

Créez la fonction `couplage_approximatif(list[str], list[str], float) -> (boolean, float)` qui :

- prend en paramètre 2 enregistrements et un seuil (nombre flottant)
- normalise le seuil entre 0 et 1.
- normalise les enregistrements à l'aide de la fonction `normaliser_champ()` ;
- calcule le score de couplage approximatif entre les enregistrements normalisés ;
- détermine s'il y *match* à l'aide du seuil et imprime "MATCH" ou "NON MATCH" selon la décision ;
- renvoie un tuple contenant 2 valeurs : (1) un booléen donnant la décision de couplage et (2) le score de similarité entre les deux enregistrements.

Testez votre méthode de couplage approximative sur les paires d'enregistrements suivants données dans `couplage.py` pour cette question.
Testez différentes valeurs de seuil pour la fonction `couplage` : trouvez-vous facile de déterminer une valeur satisfaisante pour tous les cas ?

Tip

🧪 **Sortie attendue**

```
Couplage
Paramètres seuil= 0.5
✅ MATCH (0.84) :: ('Lanet (Mme J.-A', 'Lanet (Mme)') • ('professeur d" "harmo
✅ MATCH (1.00) :: ('Laplace et Dumont (Mlles', 'Laplace et Dumont (Mlles)') •
✅ MATCH (0.74) :: ('Regnault et vue Poupinel', 'Regnault et Vve Poupinel') •
✅ MATCH (0.50) :: ('Dasté', 'Dasté (J.)') • ('menuisier-ébèniste', 'menuisier
⛔ NON MATCH (0.47) :: ('Goix (Mme.)', 'Goix') • ('dentelière', 'dentelle et a
```

Note

📝 **À retenir.**
Lorsque les enregistrements à coupler contiennent des erreurs, ou de petites différences, le couplage exact est mis en échec et génère des **faux négatifs**, c'est-à-dire des couplages qui n'ont pas été identifiés. Dans ce type de situation, on essaye plutôt de construire une mesure de la ressemblance entre deux enregistrements. Lorsque les champs sont des chaînes de caractères, on utilise des mesures de similarité ou de distance entre mots pour cela. Il faut alors se doter d'une méthode de décision pour choisir si une valeur de similarité / distance entre deux enregistrements signifie qu'ils sont couplés, ou non.

D/ Un processus minimaliste mais complet de couplage

d'enregistrements

Nous avons jusqu'ici :

- vu l'importance de pré-traiter les chaînes de caractère pour faciliter leur comparaison ;
- testé deux techniques de couplage déterministes, l'une exacte adaptée aux enregistrements sans erreurs, et l'autre approximative plus souple mais dont les résultats sont moins facilement interprétables ;
- testé une méthode simple de classification d'une paire d'enregistrements en *match* ou *non-match* utilisant un seuil de similarité.

Les techniques implémentées sont relativement simples et il existe des approches de couplage extrêmement raffinées.
Toutefois, la structure d'une méthode de couplage déterministe entre deux tables A et B

suit généralement quatre étapes :

1. **Pré-traitements** : Normaliser, éliminer les enregistrements vides, etc.
2. **Comparaison** : Créer toutes les paires possibles d'enregistrements de A et B. Si

   chaque table contient 100 enregistrements, il faut créer 10 000 paires !
3. **Classification** : Décider pour chaque paire si c'est un *match* ou un *non match*
4. **Post-traitements** : Supprimer toutes les paires *non match*, et renvoyer les *matches*.

> Important
>
> ✏️ **- QUESTION 10 - ⭐⭐⭐⭐**
>
> ✏️ **Q7**. Implémentez la fonction `couplage(list[list[str]], list[list[str]], float )-> list[list[]]` qui :
>
> - prend en paramètre deux listes d'enregistrements A et B (deux listes de listes de `str`, donc.), et un seuil de couplage ;
>   initialise une liste de couplages vide `liste_couplages` ;
>   itère sur toutes les paires possibles d'enregistrements de A et B et applique `couplage_approximatif()` pour la classer la paire en *match* ou *non match*. S'il y a *match*, ajoute un tuple `(i, j, score)` dans liste_couplages où
>   - `i` et `j` sont les indices dans A et B respectivements des deux enregistrements couplés
>   - `score` est le score de couplage;
> - retourne `liste_couplages`

Finalement, testez votre méthode de couplage avec les deux listes d'enregistrements données dans pour cette question `couplage.py`.

> 💡 Tip
>
> **🧪 Sortie attendue**

```
Couplage
Paramètres seuil= 0.3 alpha= 1.0
table_1[0] & table_2[0] ⇨ ✅ MATCH (0.50) :: ('Dasté', 'Dasté (J.)') • ('menu
table_1[2] & table_2[1] ⇨ ✅ MATCH (0.84) :: ('Lanet (Mme J.-A', 'Lanet (Mme)
table_1[3] & table_2[2] ⇨ ✅ MATCH (1.00) :: ('Laplace et Dumont (Mlles', 'La
table_1[4] & table_2[3] ⇨ ✅ MATCH (0.74) :: ('Regnault et vue Poupinel', 'Re

[🧪 Bonus si tabulate est installé !]
+--------------------------+--------------------------+------------------+-----
| 1.PER                    | 1.ACT                    | 1.LOC            | Matc
+==========================+==========================+==================+=====
| Dasté                    | menuisier-ébèniste       | Notre-Dame, 27   | ✅
| Loze (Jean)              | potier d'etain           | Greneta 12       | ⛔
| Lanet (Mme J.-A          | professeur d" "harmonie  | Beaux-Arts 6     | ✅
| Laplace et Dumont (Mlles | institutrices            | Lions-St-Paul 14 | ✅
| Regnault et vue Poupinel | fab. douates, depĕt      | 47 Cha-ronne     | ✅
|                          |                          |                  | ⛔
+--------------------------+--------------------------+------------------+-----
```

Ouf, c'est fini ! 🏁

C'est tout pour cette fois, vous voici arrivé(e)s au bout, félicitations ! 🎉🎉

Dans la prochaine séquence, nous apprendrons à transformer la méthode de couplage implémentée en un outil **en ligne de commande** réutilisable !