

Tutoriel

LLM · Qualité des données · Pandas

Nettoyer un CSV avec l'aide d'un LLM

Détecter les anomalies textuelles d'un petit CSV, demander des propositions au modèle et appliquer uniquement des corrections vérifiables.

Auteur **Youness Boutyour**

**DURÉE**

60 minutes

NIVEAU

Intermédiaire

FORMAT

Tutoriel pratique

LLM · Qualité des données · Pandas · publié le 2026-06-12 · mock

Objectifs et livrables

Un export CSV peut contenir « Casa », « Casablanca », des espaces superflus et plusieurs écritures d'une même catégorie. Nous allons utiliser un LLM pour proposer des rapprochements, puis nous appuyer sur un référentiel et des règles Python pour décider des corrections à appliquer.

Vous commencerez par préparer le fichier bruité et les valeurs de référence. Nous demanderons ensuite des propositions structurées au modèle, vérifierons leur conformité et appliquerons les changements acceptés. Les valeurs ambiguës permettront de voir comment le programme signale une ligne à revoir.

Ce parcours rend chaque décision explicable : quelle valeur a été reçue, quelle correction a été proposée et pourquoi a-t-elle été retenue ou refusée ? Nous examinerons la trace des changements en parallèle du fichier nettoyé, afin de pouvoir comparer le résultat aux données de départ.

À la fin, vous aurez `outputs/clean.csv`, `outputs/changes.jsonl` et un compteur de lignes à examiner. Ces fichiers vous donneront une base pour vérifier la qualité du nettoyage et faire évoluer les règles avec votre propre référentiel.

Utiliser le LLM comme proposeur, pas comme arbitre

Schéma du nettoyage CSV qui confronte une proposition du LLM à un référentiel et conserve la trace de chaque correction.



Les valeurs ambiguës restent visibles dans la trace au lieu d'être corrigées silencieusement.

Lire le flux avant de coder

Le schéma positionne le LLM comme un proposeur de correspondances. Le référentiel contient les valeurs autorisées et le code Python décide si une proposition est suffisamment sûre pour être appliquée. Les valeurs ambiguës sont conservées dans le journal plutôt que forcées vers une catégorie plausible.

Pour chaque correction, comparez la valeur d'origine, la proposition, la règle utilisée et le résultat écrit dans `clean.csv`. Cette lecture rend le nettoyage réversible et permet de corriger le référentiel sans perdre les décisions déjà prises.

1. Préparer l'environnement

Séparez dès le départ les fichiers d'entrée, les propositions du modèle et les données exportées. Le CSV original doit rester le point de comparaison : une nouvelle exécution pourra ainsi être relue sans perdre les valeurs qui ont motivé les corrections. Installez les dépendances dans l'environnement virtuel, puis exécutez les commandes depuis la racine du projet.

Le chargement avec `dtype=str` conserve les cellules sous forme de chaînes, ce qui évite une conversion implicite des libellés ou d'éventuels identifiants numériques. `fillna("")` remplace les valeurs manquantes par une chaîne vide ; cela facilite les comparaisons, mais ne constitue pas une correction métier. Une cellule vide restera à examiner si aucune règle ne permet de la compléter.

Code de l'étape

Terminal · Bash

```
mkdir -p llm-nettoyage-csv/{data,src,outputs}
cd llm-nettoyage-csv
python3 -m venv .venv && source .venv/bin/activate
python -m pip install pandas
```

Sortie obtenue

Sortie / exemple

Successfully installed pandas

Commande de vérification

Terminal · Bash

```
python -c "import pandas as pd; print(pd.__version__)"
```

2. Créer le CSV bruité et les règles

Le référentiel définit les libellés vers lesquels une correction peut converger. Les variantes orthographiques servent à tester les rapprochements simples ; les cas ambigus et la ville absente du référentiel permettent de vérifier que le programme n'impose pas une réponse arbitraire. Parcourez ces lignes avec les listes de villes et de catégories avant d'interroger le modèle.

Conservez une distinction entre une valeur incorrecte et une valeur inconnue. Une nouvelle ville peut être légitime même si elle ne figure pas encore dans la liste autorisée. Le tutoriel l'oriente vers la revue plutôt que d'enrichir automatiquement le référentiel. Cette séparation rend le résultat plus utile : on peut améliorer les données sans transformer chaque suggestion du modèle en règle.

Créez `data/contacts.csv` avec seize lignes et les colonnes `name,city,category`. Ajoutez `data/reference.json` avec trois villes et trois catégories autorisées. Le jeu contient des variations de casse, des espaces parasites, une nouvelle ville valide et deux valeurs réellement ambiguës.

Extrait de `data/contacts.csv`

Le tableau ci-dessous reprend quelques lignes du fichier source. Le symbole `SP` représente un espace final volontairement conservé pour rendre l'anomalie visible dans l'aperçu.

Nom	Ville	Catégorie
Amina	Rabat	student
Youssef	Casablanca	review
Sara	rabat	student
Omar	Casablanca SP	review
Nadia	Rabat	student
Mehdi	Casablanca	review

Cet extrait permet de repérer immédiatement les écarts de casse et les espaces parasites avant l'étape de normalisation.

Sortie obtenue

Le jeu contient six variations volontaires, une nouvelle ville et deux lignes ambiguës.

■ Sortie / exemple

rows=16 columns=3 variations=6 valid_new_values=2 ambiguous_rows=2

Commande de vérification

Terminal · Bash

```
python -c "import pandas as pd; print(pd.read_csv('data/contacts.csv').shape)"
```

Code de l'étape

Python

```
import json
import pandas as pd

rows = pd.read_csv("data/contacts.csv", dtype=str).fillna("")
reference = json.loads(open("data/reference.json", encoding="utf-8").read())
print(rows.shape, reference.keys())
```

3. Demander une proposition structurée

Le modèle propose des opérations ; il ne reçoit pas l'autorisation de modifier directement le tableau. Chaque proposition associe un numéro de ligne, un champ, une ancienne valeur et une nouvelle valeur. Cette forme permet au programme de contrôler précisément la modification, au lieu de remplacer le CSV par une version entièrement réécrite dont les différences seraient difficiles à expliquer.

Le champ `confidence` est une estimation fournie dans la proposition, pas une probabilité calibrée de correction. Il sert ici à appliquer un seuil de tri, en complément des règles locales. Exécutez d'abord le fichier de propositions mock pour comprendre les refus, puis comparez les propositions réelles aux mêmes lignes sans modifier simultanément le seuil ou le référentiel.

Le prompt doit demander une liste JSON de corrections avec `row_id`, `field`, `old`, `new`, `confidence`. En mode mock, utilisez les corrections de `examples/mock-corrections.json`.

Sortie obtenue

Chaque proposition indique la ligne, l'ancienne valeur et la nouvelle valeur ; aucune modification n'est encore appliquée.

Sortie / exemple

```
proposals=4 applied=0 review=4 output=outputs/proposals.json
```

Le mode réel utilise `src/llm_client.py` et écrit la proposition brute dans `outputs/proposals.json`. Les règles locales restent obligatoires avant toute application.

Code de l'étape

```
Python

import json
from llm_client import chat

prompt = (
    "Retourne uniquement une liste JSON de corrections avec "
    "row_id, field, old, new et confidence.\n"
    + rows.to_json(orient="records", force_ascii=False)
)
raw = chat([{"role": "user", "content": prompt}])
proposals = json.loads(raw)
open("outputs/proposals.json", "w", encoding="utf-8").write(raw)
```

Commande de vérification

```
Terminal · Bash

python src/clean.py --mock --input data/contacts.csv --proposals examples/mock-corrections.json
```

4. Valider, appliquer et auditer

L'application commence par vérifier que la proposition vise toujours la valeur observée. `row_id` est numéroté à partir de 1, alors que `iloc` commence à 0 : la soustraction de 1 relie ces deux conventions. La comparaison avec `old` évite d'appliquer une correction périmée si le fichier a changé entre la proposition et son traitement.

Lisez les exports ensemble : le CSV contient les valeurs retenues, `changes.jsonl` décrit les opérations appliquées et `review.jsonl` conserve les situations non résolues automatiquement. Le nombre de corrections n'est pas nécessairement le nombre de lignes modifiées, car une ligne peut recevoir plusieurs opérations. Comparez enfin le CSV exporté au fichier source, notamment les lignes qui ne devaient pas changer.

Appliquez une correction seulement si `new` appartient à la référence et si `confidence` ≥ 0.9 . Écrivez les propositions refusées dans `outputs/review.jsonl`.

Sortie obtenue

Le CSV nettoyé respecte les listes de référence et chaque changement est traçable.

Sortie / exemple

```
rows=16 clean=14 accepted=5 review=2 invalid_values=0 changes_log=outputs/changes.jsonl
```

Commande de vérification

Terminal · Bash

```
python src/clean.py --mock --check --input data/contacts.csv --proposals examples/mock-corrections.json
python src/quality.py outputs/clean.csv data/reference.json
```

Code de l'étape

Python

```
changes, review = [], []
for proposal in proposals:
    row = rows.iloc[proposal["row_id"] - 1]
    accepted = (
        proposal["field"] in reference and
        row[proposal["field"]] == proposal["old"] and
        proposal["new"] in reference[proposal["field"]] and
        float(proposal["confidence"]) ≥ 0.9
    )
    (changes if accepted else review).append(proposal)
    if accepted:
        rows.loc[proposal["row_id"] - 1, proposal["field"]] = proposal["new"]
rows.to_csv("outputs/clean.csv", index=False)
```

Exemples de sorties

Le mode mock produit un CSV valide, un journal des changements et au moins une ligne en revue humaine. Une correction silencieuse est considérée comme un échec.

Dépannage et diagnostics

Si le modèle renvoie du Markdown autour du JSON, refusez la réponse ou utilisez une sortie structurée. Conservez le prompt, le modèle et le hash du fichier source.

Vérification finale

- ☐ seize lignes traitées ;
- ☐ valeurs finales dans les référentiels ;
- ☐ modifications et refus journalisés ;
- ☐ ligne ambiguë envoyée en revue.

Limites et prolongements

Le LLM ne remplace pas les règles métier. Pour un volume important, échantillonnez les corrections et mesurez le taux d'acceptation par colonne.

Références

- [pandas — read_csv](#)
- [Great Expectations — documentation](#)