Projet Elexir

# ✅ To-Do : Projet “Bibliothèque de Data Engineering”

* 📦 \*\*Structurer un repo GitHub\*\* pour la bibliothèque (`dataeng\_toolkit` ou autre nom sympa)
* 🧱 Créer les premiers modules :
  + `text\_utils.py` → nettoyage de texte chaînable
  + `df\_utils.py` → gestion de concaténation, renommage, fusion
* 📄 Rédiger un `README.md` clair :
  + Objectifs de la bibliothèque
  + Exemples d’utilisation (chaînage des transformations, pipelines)
* 🧪 Mettre en place les premiers tests (`pytest`)
* 📁 Ajouter un `pyproject.toml` pour la distribution `pip`
* 💡 Bonus : un `notebook\_demo.ipynb` avec un use case clair

Exemple

1. import datalixir as dx

2.

3. clean\_df = (

4. dx.from\_df(df)

5. .text('description').strip().lower().remove\_accents().result()

6. .text('commentaire').strip().replace\_words({'bad': 'poor'})

7. .export()

8. )

9.

# Contenu de départ

## 🧱 Ce que pourrait contenir la bibliothèque

- `text\_utils.py` → nettoyage de texte (comme on a commencé)

- `df\_utils.py` → concaténation, renommage, sélection intelligente

- `pipeline.py` → enchaînement de transformations façon `sklearn`

- `validation.py` → fonctions pour tester la cohérence ou la qualité des données

- `io\_helpers.py` → chargement/sauvegarde intelligents (avec options typage, cache, etc.)

## 🛠️ Structure de repo GitHub recommandée

dataeng\_toolkit/

├── dataeng\_toolkit/

│ ├── \_\_init\_\_.py

│ ├── text\_utils.py

│ ├── df\_utils.py

│ ├── pipeline.py

│ └── ...

├── tests/

│ ├── test\_text\_utils.py

│ └── ...

├── setup.py

├── pyproject.toml

├── README.md

└── examples/

└── notebook\_demo.ipynb

* Avec `pytest`, `ruff` ou `black` pour le linting & la qualité du code
* Un logo sympa peut-être 😎

## 📦 Distribution via `pip`

Facile : avec `pyproject.toml` (nouveau standard) ou `setup.py`, puis publication sur [[PyPI](https://pypi.org/)].

## 💡 Le petit plus

Accompagner la lib d’un ou deux notebooks “exemples de pipelines” réels :

`pipeline\_text\_classification.ipynb`

`pipeline\_cleaning\_for\_model.ipynb`

# 🔍 Ce qui existe déjà

Il existe plusieurs bibliothèques Python qui proposent des outils de nettoyage de données, mais aucune ne combine exactement ce qui est envisagé : une API fluide, chaînable, orientée text/data cleaning avec une syntaxe élégante façon `df.text('col').strip().lower().remove\_accents()`.

| Bibliothèque | Ce qu’elle fait bien | Limites pour ton cas |
| --- | --- | --- |
| `pandas` | Nettoyage de base avec `.str`, `.fillna()`, etc. | Pas d’API chaînable personnalisée |
| `pyjanitor` | Ajoute des méthodes à Pandas (`clean\_names()`, etc.) | Très orienté tabulaire, pas textuel |
| `clean-text` | Nettoyage de texte (emoji, accents, etc.) | Ne s’intègre pas à Pandas, pas chaînable |
| `textacy` | Prétraitement NLP avancé | Plutôt orienté NLP, pas Pandas |
| `sklearn.pipeline` | Pipelines de transformation | Trop généraliste pour du nettoyage ciblé |
| `dataprep` | Explorations + nettoyage semi-automatisé | Pas conçu pour du chaînage manuel fluide |

👉 Aucune ne propose une mini-DSL (langage fluide) pour manipuler des colonnes de texte dans un DataFrame avec une syntaxe aussi naturelle que celle imaginée.

# 🧪 Ce que `datalixir` apporterait de nouveau

* Une API expressive et intuitive pour le nettoyage de texte
* Une intégration fluide avec Pandas
* Une chaînabilité élégante façon `.strip().lower().remove\_accents()`
* Des outils pour concaténer, tracer, auditer les transformations
* Une philosophie modulaire : tu ajoutes tes propres briques facilement

On serait donc clairement en terrain innovant, avec un vrai positionnement entre ergonomie, lisibilité et puissance.