Outils pour la Data Science : application à la médecine personnalisée

Marie PERRIN

Post-doctorante (UGA/CHUGA/TIMC/LIG)

Meetup Python Grenoble – 16 décembre 2025

DataFrames for the new era => pour un data processing performant !

- Librairie : écriture en Rust, compatible Python
- Rapide : opérations de calcul effectuées en parallèle
- Facile d'utilisation : expressions intuitives pour un code lisible
- Open source : communauté active, licence du MIT

(https://pola.rs/, Python API reference = https://docs.pola.rs/api/python/stable/reference/index.html)

```
import polars as pl
import plotly.express as px
```

1) Comparaison Polars vs. Pandas (ouverture-sauvegarde dataframe)

```
df_patient = (
    # loading data from the csv file (with header, comma as separator, automated recognition of dates)
    pl.read_csv(
        source='./original_export_15_avril_2025/patient_did.csv',
        has_header=True,
        separator=',',
        try_parse_dates=True,
    )
```

```
from pathlib import Path

# converting the string path to a Path object
path = Path("./materials/1-patient_did_cleaned.csv")

# writing the DataFrame to CSV
df_patient.write_csv(path, separator=',')
```

Polars : pl.read_csv
Pandas : pd.read_csv

Polars : df.write_csv
Pandas : df.to_csv

```
df_acte_cnet.describe()
```

shape: (9, 7)

statistic	date_acte
str	str
"count"	"26782"
"null_count"	"0"
"mean"	"2020-07-29 15:08:53.104323"
"std"	null
"min"	"2005-01-27"
"25%"	"2019-07-12"
"50%"	"2021-04-09"
"75%"	"2022-12-16"
"max"	"2025-03-27"

2) Comparaison Polars vs. Pandas (vitesse de lecture : fichier csv de 1 million de lignes x 35 colonnes, 450 Mo)

```
df_biologie = (
    pl.read_csv(
        source='./original_export_15_avril_2025/export_document_bio_janv25.csv',
        has_header=True,
        separator=',',
        try_parse_dates=True,
        ignore_errors=True
    )
)

df_biologie
```

✓ 1.2s Open 'df_biologie' in Data Wrangler

```
df_biologie = pd.read_csv(
    filepath_or_buffer='./original_export_15_avril_2025/export_document_bio_janv25.csv',
    sep=',',
    header=0, # column names are inferred from the first line of the file,
)

df_biologie
```

✓ 8.1s Open 'df_biologie' in Data Wrangler

3) Comparaison Polars vs. Pandas (édition dataframe)

```
# renaming 'date' column into 'date_acte'
.rename({'date': 'date_acte'})
```

Polars : df.rename({'y':'year'})
Pandas : df.rename(columns = {'y':'year'})

```
# removing the hash_ipp for 'abc' =>
.remove(pl.col('hash_ipp') == '&bc')
```

Polars : df.remove(pl.col('col') == 'abc') => .filter, etc...
Pandas : df.query('col' == 'abc'})

```
    # 1st level of CCAM classification = blocs (X)
    filtered_df_pre_lam_unique_clean_2 = (
        filtered_df_pre_lam_unique_clean.drop('ccam_libelle_long_FR')
        .join(ccam_code_matching, on='ccam_code_slice', how='left')
        .select('hash_ipp', 'hash_ven_num', 'date_admission', 'date_a
                'ccam_code_slice', 'ccam_libelle_long_FR', 'date_
        )
    )

    filtered_df_pre_lam_unique_clean_2
```

Polars : df.drop('col')
Pandas : df.drop(columns=['abc'])

Polars : df1.join(df2, on='col', how='left')
Pandas : pd.merge(df1, df2, how='left', on='col'))

Polars : df.select('a', 'b', 'c', 'd')
Pandas : df[['a', 'b', 'c', 'd']]

```
# changing the date_diagnostic_lam_de_novo for 'abc'
.with_columns(
    pl.when(pl.col('hash_ipp') == 'abc')
    .then(pl.lit(date(2019, 10, 25)))
    .otherwise(pl.col('date_diagnostic_lam_de_novo'))
    .alias('date_diagnostic_lam_de_novo')
)
```

Pour le patient 'abc'
Change la date de diagnostic (25-10-2019)
En écrasant la date existante

Sinon, ne change rien !

.with_columns (ajout de colonne, remplacement du contenu d'une colonne existante)

```
# regex: replacing 'NUTRISON ENERGY 1.5 NUTRICIA 500ML' by 'NUTRISON ENERGY NUTRICIA 500ML'
df = df.with_columns(
    libelle=pl.col('libelle').replace('NUTRISON ENERGY 1.5 NUTRICIA 500ML', 'NUTRISON ENERGY NUTRICIA 500ML')
)
```

Polars : replace=pl.col('col').replace('a', 'b')
Pandas :
df['col'].str.replace('a', 'b', regex=True)

4) Comparaison Polars vs. Pandas (fonctions avancées)

Polars : pl.concat([df1, df2])
Pandas : pd.concat([df1, df2])

Polars : df.unique(subset=['col1', 'col2'], keep='first')
Pandas : df.drop_duplicates(subset=['col1', 'col2'], keep='first')

```python
# counts of unique diags per ccam_code_slice (main category)
clean_ccam_code_slice_acts = (
    filtered_df_pre_lam_unique_clean_unique.group_by(
        'ccam_code_slice', maintain_order=True).agg(pl.count('ccam_code_slice').alias('nombre_actes_uniques_par_catégorie')
    )
)

clean_ccam_code_slice_acts = clean_ccam_code_slice_acts.sort('nombre_actes_uniques_par_catégorie', descending=True)
clean_ccam_code_slice_acts
```

shape: (7, 2)

ccam_code_slice	nombre_actes_uniques_par_catégorie
str	u32
"Z"	738
"D"	379
"E"	263
"G"	174
"F"	102
"J"	85
"H"	43

Projet AI4DigAML

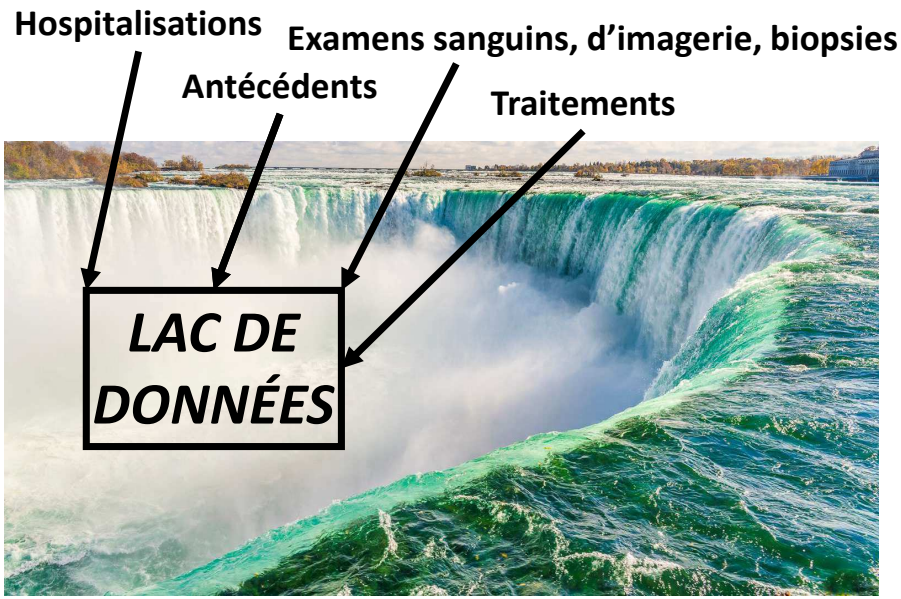
L'IA pour identifier les facteurs prédictifs de la LAM (Leucémie Aigüe Myéloïde)

- Addition d'erreurs dans les cellules souches du sang
- Moyenne d'âge (62 ans), taux de survie à 5 ans (35%)
- Phase pré-cancéreuse « masquée » = diagnostic tardif

Étude des trajectoires de patients « malades » vs. « témoins »

Identifier les évènements de santé spécifiques qui précèdent la LAM

315 patients diagnostiqués (2018-2024)



Objectifs du projet

Pour les patients identifiés à haut risque de développer une LAM,
proposer un suivi préventif, avec une évaluation régulière et complète de leur état de santé

1) Table de contigence, matrice de comptage (df.write_csv dans Polars => pd.read_csv dans Pandas)

```python
# contigency table

df_ipp_lib_count_pivot = (
    df_ipp_lib_count.pivot(on='libelle_long', index='hash_ipp', aggregate_function='sum')
    .fill_null(strategy="zero")
)

df_ipp_lib_count_pivot
```

Polars : pas de concept 'index' comme 1ère colonne de référence du dataframe
Pandas : l'index (index_col='hash_ipp') doit toujours être mis en place après la conversion

=> Apache Arrow fonctionne aussi, mais son implémentation pose des problèmes plus loin…

```python
df_lib_count_pivot_pd.columns.name = 'modality'
df_lib_count_pivot_pd
```

✓ 0.0s Open 'df_lib_count_pivot_pd' in Data Wrangler

Pandas

modalité	Acte 1	Diag 1	Drug 1	Acte 2	Diag 2	Drug 2	Acte 3	Diag 3	Drug 3	…
hash_ipp										
ABC123	2	**0**	4	**0**	**0**	1	**0**	2	**0**	…
EFG456	**0**	5	2	**0**	1	3	**0**	**0**	1	…
HIJ789	5	22	16	2	5	11	12	11	2	…
…	…	…	…	…	…	…	…	…	…	…

116 patients

29 modalités
- Actes
- Diagnostics
- Traitements

2) Librairie pyPLNmodels

https://pypi.org/project/pyPLNmodels/

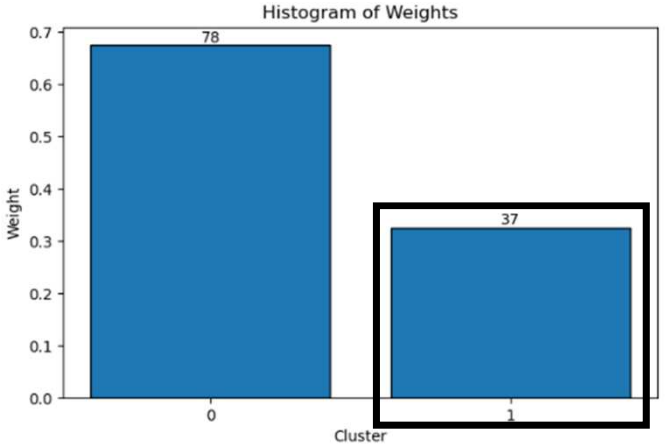
- Modèle de Poisson log-normal => analyse statistique de matrices de comptage multi-dimensionnelles (distribution non-gaussienne)
- Clustering non-supervisé (Mixture)
- Réduction de dimension (PCA)
- Zero-inflation (ZI)
- Inférence de réseau (Network)

Consider $\mathbf{Y}$ a count matrix (denoted as "endog" in the package) consisting of n rows (here, patients) and p columns (here, modalities)

- Each individual $\mathbf{Y}_i$, that is the i^{th} row of $\mathbf{Y}$, is independent from the others. It follows a Poisson lognormal distribution (PLN-equation) : $\mathbf{Y}_i \sim \mathcal{P}(exp(\mathbf{Z}_i)), \mathbf{Z}_i \sim \mathcal{N}(\mathbf{o}_i + \mathbf{B}^T\mathbf{x}_i, \mathbf{\Sigma})$
- $\mathcal{P}$ (respectively $\mathcal{N}$) denotes a Poisson (respectively Normal) distribution
- $\mathbf{Z}_i$, the latent variables, are not directly observable. However, from a statistical perspective, they provide more informative insights compared to the observed variables $\mathbf{Y}_i$
- $\mathbf{Z}$ is a $n \times p$ matrix of unobserved latent Gaussian vectors $\mathbf{Z}_i$: the latent layer $\mathbf{Z}$ is not observed

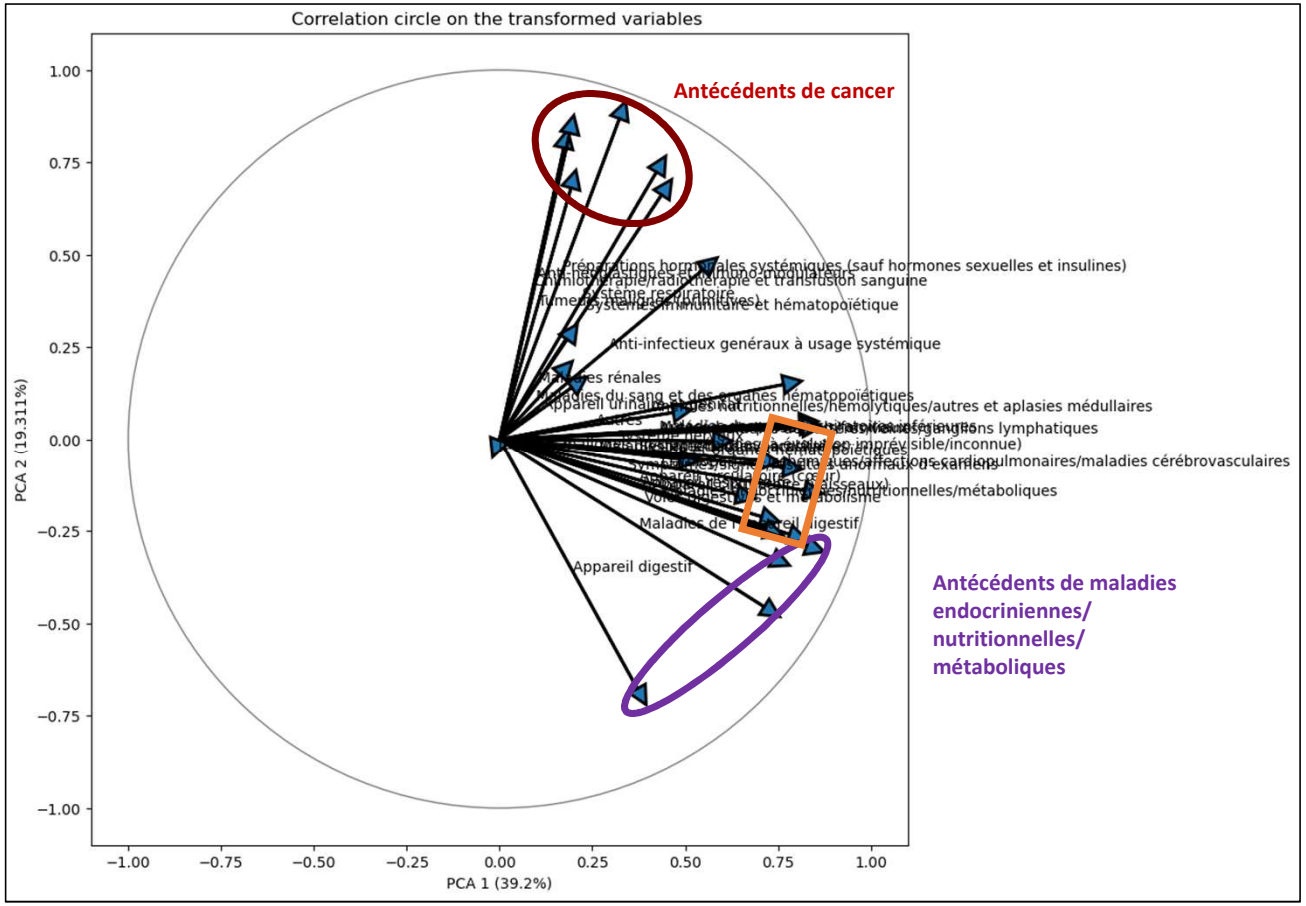
Estimating the unknown parameter $\mathbf{\Sigma}$, denoted as "covariance"

3) PlnMixture (clustering non-supervisé)



4) ZIPlnPCA (Zero-Inflation + Principal Component Analysis)

- Zero-inflation : adapté aux données du monde réel (>50% zéros dans la matrice), ex. données de écologie/biologie/médecine
- Principal Component Analysis : projection des 29 modalités dans un repère en 2 dimensions



Les 2 axes de l'ACP (Analyse en Composantes Principales) représentent une forte proportion (58.3%) de l'analyse
=> ACP fiable

Récapitulatif

API en Python (Polars vs. Pandas)

Pandas (https://pandas.pydata.org/)

- Une des bibliothèques DataFrame les plus utilisées
- À privilégier quand on débute en Data Science (apprentissage des opérations de formatage, regroupement, résumé des données en dataframes)
- Difficultés avec de très grands ensembles de données (contraintes de mémoire)

Polars (https://pola.rs/)

- Bibliothèque DataFrame récente, conçue pour effectuer des opérations sur les données, efficaces et parallélisées
- Écrite en Rust pour une meilleure performance (gestion et vitesse de la mémoire de bas niveau)
- Excellente gestion des grands ensembles de données (pas de contraintes de mémoire)

Propriétés communes aux écosystèmes Pandas et Polars

- Facilité d'utilisation (noms intuitifs des fonctions)
- Syntaxe similaire (nécessite un temps d'adaptation lors du passage de Pandas à Polars)
- Lecture/écriture de nombreux formats de fichiers (CSV, Excel, SQL, JSON, HDF5, etc.)
- Intégration facile d'autres librairies (NumPy, SciPy, Matplotlib, Plotly, Scikit-learn, etc.)
- Prise en charge efficace des données manquantes (fonctions dédiées)
- Inter-opérabilité des dataframes (outil de formatage de données "Apache Arrow")

Projet IA4DigAML (IA pour identifier les facteurs prédictifs de la LAM = Leucémie Aigüe Myéloïde)

- Table de contigence = matrice de comptage (116 patients x 29 modalités)
- Modalités : actes/diagnostics/traitements médicaux effectués au CHU de Grenoble, dans les années précédent le diagnostic de LAM
- Librairie pyPLNmodels (https://pypi.org/project/pyPLNmodels/) basée sur un modèle mathématique de Poisson log-normal

Résultats préliminaires : les patients diagnostiqués pour une LAM, présentent en particulier les antécédents médicaux suivants :

- cancers, maladies cardiovasculaires, maladies endocriniennes/nutritionnelles/métaboliques (ex. diabète)