

Introduction au Machine Learning



Séance 4 — Des données propres : préparer avant d'apprendre

Dr. El Hadji Bassirou TOURÉ

Département de Mathématiques et Informatique, Faculté des Sciences et Techniques, Université Cheikh Anta Diop de Dakar

<https://ebtoure.sn/cours/introduction-machine-learning/chapitres/04-donnees-propres.html>

Version du 11 octobre 2026



Version en ligne

Introduction au ML — Séance 4

Des données propres : préparer avant d'apprendre

Dr. El Hadji Bassirou TOURÉ

Département de Mathématiques et Informatique, Faculté des Sciences et Techniques, Université Cheikh Anta Diop de Dakar

Objectifs de la séance

Contenu

La Séance 3 a entraîné un arbre sur des données impeccables. Cette séance affronte les **données réelles** — trouées, hétérogènes, piégées — sur un vrai jeu médical (Pima), et construit la chaîne de préparation rigoureuse qui rend un modèle digne de confiance.

- Détecter et **réparer les valeurs manquantes** : taux de manquants, imputation par la médiane.
- **Encoder** les catégories (one-hot) et **mettre à l'échelle** les variables (z-score).
- Un second modèle, les k **plus proches voisins** — pour qui l'échelle est une condition de validité.
- Le danger central du métier : la **fuite de données** — et le **Pipeline** qui l'empêche,
 $f = m \circ \varphi$.

Des données réelles, enfin

Le réveil brutal

Changement de décor

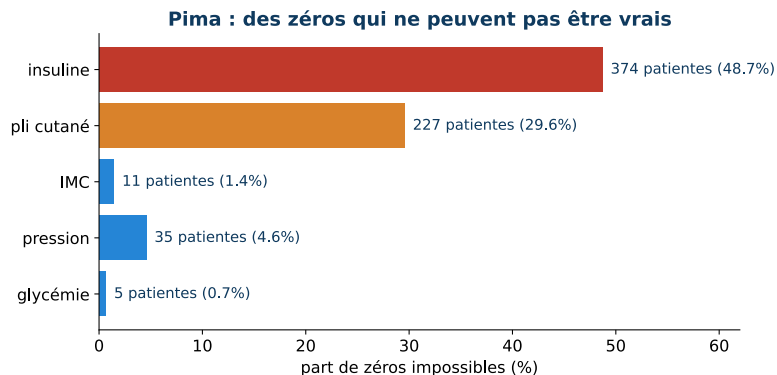
DataSANTÉ-221 était un registre impeccable : aucune case vide, des échelles raisonnables. Les données réelles ne ressemblent jamais à cela. Cette séance travaille sur **Pima** — une vraie étude médicale : 768 patientes, 8 mesures cliniques, et une cible : le **diabète** (268 positives, 500 négatives).

Les trois premières patientes — extrait

patiente	glycémie	pression	insuline	IMC	âge	diabète
x_1	148	72	0	33,6	50	$y_1 = 1$
x_2	85	66	0	26,6	31	$y_2 = 0$
x_3	183	64	0	23,3	32	$y_3 = 1$

Une ligne par patiente, comme toujours. Mais un détail cloche déjà : trois insulines à 0...

Le problème caché : des zéros impossibles



Une glycémie de 0 ou une pression artérielle de 0 sont *physiologiquement impossibles* : ce ne sont pas des mesures, ce sont des **absences de mesure** codées par un 0. Près d'une patiente sur deux n'a pas de dosage d'insuline (374/768). Le fichier ne ment pas — il *cache* ses trous.

La valeur manquante, formellement

Définition

Une **valeur manquante** est une case du tableau pour laquelle aucune mesure n'existe. Elle se note **NaN** (*Not a Number*) — jamais 0, qui est une valeur comme une autre. Le **taux de manquants** d'une colonne mesure l'ampleur du trou.

$$m_j = \frac{1}{n} \sum_{i=1}^n \mathbf{1}[x_{ij} \text{ manquant}]$$

La somme compte les cases vides de la colonne j ; la division par n en donne la proportion — exactement la mécanique du risque empirique de la Séance 3, appliquée aux trous plutôt qu'aux erreurs.

Exemple — les taux de manquants de Pima

Après remplacement des zéros impossibles par NaN

colonne	glycémie	pression	IMC	pli cutané	insuline
manquants	5	35	11	227	374
m_j	0,007	0,046	0,014	0,296	0,487

Pour l'insuline : $m = \frac{374}{768} = 0,487$ — près d'une case sur deux.

Cinq colonnes touchées, à des degrés très différents : la réparation devra être *automatique et colonne par colonne*.

Exercice — calculer un taux de manquants

Énoncé

La colonne pression compte 35 valeurs manquantes sur 768 patientes. **(a)** Calculer m_{pression} .
(b) Si l'on jetait toute ligne ayant *au moins un* trou, il resterait 392 patientes : quelle proportion du jeu perdrait-on ?

Correction

(a) $m = \frac{35}{768} = 0,046$, soit 4,6 %. **(b)** On perdrait $768 - 392 = 376$ patientes, soit $\frac{376}{768} = 49\%$ du jeu — la moitié des données sacrifiée pour des trous réparables.

Piège — entraîner sur les zéros tels quels

Erreur fréquente

Laisser les zéros impossibles dans les données. Le modèle apprend alors qu'il existe des patientes « à glycémie nulle » — une population fantôme — et place des coupures absurdes autour de 0. Le score peut même sembler correct : le poison est *silencieux*.

Erreur fréquente — bis

Jeter toutes les lignes incomplètes : ici, 49 % des patientes disparaîtraient. Pire, elles ne disparaissent pas *au hasard* — les patientes sans dosage d'insuline sont souvent celles des structures les moins équipées : on biaiserait l'échantillon en croyant le nettoyer.

La voie raisonnable : **garder les lignes, remplir les trous**. C'est l'imputation.

Réparer : l'imputation par la médiane

L'idée : remplir avec une valeur plausible

Le principe

Imputer, c'est remplacer chaque case vide d'une colonne par une valeur *typique* de cette colonne. La question devient : quelle valeur typique ? La moyenne semble naturelle — mais elle a un défaut rédhibitoire sur les données médicales.

Analogie

Pour estimer le loyer « typique » d'un quartier, la moyenne se laisse entraîner par une seule villa de luxe ; le loyer *médian* — celui du logement du milieu — reste fidèle au quartier. Les colonnes médicales ont leurs villas de luxe : les valeurs extrêmes.

La médiane, formellement

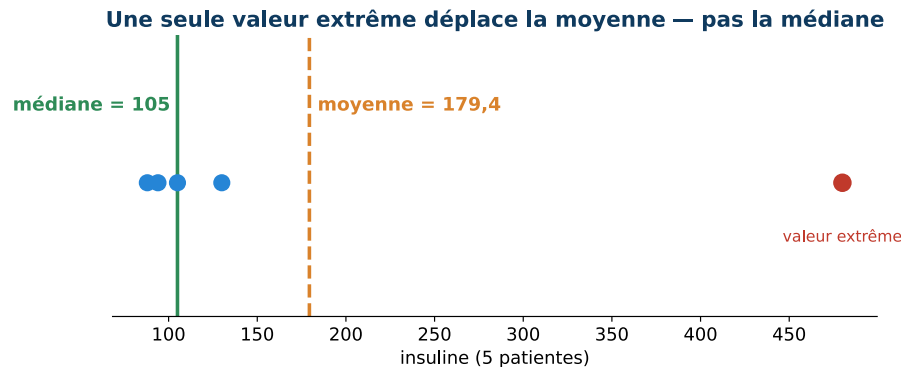
Définition

La **médiane** d'une colonne est la valeur du *milieu* une fois les valeurs triées : la moitié des valeurs lui est inférieure, la moitié supérieure. Contrairement à la moyenne, elle est **robuste** : une valeur extrême ne la déplace presque pas.

Exemple numérique — cinq insulines

Valeurs triées : 88 ; 94 ; **105** ; 130 ; 480. La médiane est la 3^e valeur : **105**. La moyenne, elle, vaut $\frac{88 + 94 + 105 + 130 + 480}{5} = 179,4$ — tirée vers le haut par la seule valeur 480, au point de dépasser 4 patientes sur 5.

La robustesse, en image



Quatre patientes entre 88 et 130, une à 480 : la moyenne (179,4) ne ressemble à *aucune* patiente ; la médiane (105) reste au cœur du groupe. C'est elle qu'on injecte dans les trous.

L'imputation, formellement

Définition

L'**imputation par la médiane** remplace chaque valeur manquante de la colonne j par la médiane med_j de cette colonne — médiane calculée sur le **jeu d'entraînement uniquement**.

$$\tilde{x}_{ij} = \begin{cases} x_{ij} & \text{si } x_{ij} \text{ est observée} \\ \text{med}_j & \text{si } x_{ij} \text{ est manquante} \end{cases}$$

Point décisif : med_j est un **paramètre appris** des données, au même titre que les coupures de l'arbre. Et comme tout paramètre appris, il s'apprend sur le train — *jamais* sur le test. La raison précise arrive à la section « fuite de données ».

Exemple — l'imputation en chiffres sur Pima

Médianes apprises sur le train

Sur les 614 patientes d'entraînement : $\text{med}_{\text{insuline}} = 125,5$, $\text{med}_{\text{glycémie}} = 118,0$, $\text{med}_{\text{IMC}} = 32,0$.

La patiente n°525 (du test) a une insuline manquante : $\tilde{x}_{525, \text{insuline}} = \text{med}_{\text{insuline}} = 125,5$.
Ses valeurs observées (glycémie 87, âge 21) restent intactes.

À comparer : la *moyenne* de l'insuline sur le train vaut 139,4 — gonflée par les valeurs extrêmes, elle aurait rempli les trous avec une valeur peu typique.

Exercice — imputer à la main

Énoncé

Une mini-colonne IMC contient : 24,1 ; NaN ; 31,5 ; 28,0 ; NaN ; 35,2. **(a)** Calculer la médiane des valeurs observées. **(b)** Donner la colonne imputée.

Correction

(a) Valeurs observées triées : 24,1 ; 28,0 ; 31,5 ; 35,2 — nombre pair, la médiane est la moyenne des deux du milieu : $\frac{28,0 + 31,5}{2} = 29,75$. **(b)** Colonne imputée : 24,1 ; **29,75** ; 31,5 ; 28,0 ; **29,75** ; 35,2 — les valeurs observées ne bougent pas.

En pratique : SimpleImputer

```
import numpy as np
from sklearn.impute import SimpleImputer

X[colonnes_impossibles] = X[colonnes_impossibles].replace(0, np.nan)
imputeur = SimpleImputer(strategy="median")
imputeur.fit(X_train) # apprend les medianes (train !)
X_train_p = imputeur.transform(X_train)
X_test_p = imputeur.transform(X_test) # applique les MEMES medianes
```

Le motif `fit` / `transform` est le cœur de toute la séance : `fit` *apprend* les paramètres (ici, une médiane par colonne), `transform` les *applique*. Le test ne voit jamais `fit` — il ne fait que subir `transform`.

Encoder les catégories

Le problème : les modèles mangent des nombres

Le constat

Distances, seuils, moyennes : tout ce que manipule un modèle est **numérique**. Or les données réelles portent des **catégories** — une région ($\{\text{Dakar, Thiès, Kaolack, Ziguinchor, Saint-Louis}\}$), un sexe, un type d'admission. Il faut les traduire en nombres *sans inventer d'information*.

Erreur fréquente

Coder Dakar = 1, Thiès = 2, ..., Saint-Louis = 5. Ces entiers fabriquent un **ordre fictif** (Saint-Louis > Dakar ?) et de fausses distances (Kaolack serait « deux fois plus loin » de Dakar que Thiès). Le modèle prendra cette géométrie inventée au sérieux.

L'encodage one-hot, formellement

Définition

L'encodage **one-hot** remplace une colonne à K modalités par K colonnes binaires : la colonne de la modalité observée vaut 1, toutes les autres 0. Aucun ordre, aucune distance fictive — chaque modalité est à égale distance de toutes les autres.

Exemple — la région en cinq colonnes

région	Dakar	Thiès	Kaolack	Ziguinchor	Saint-Louis
« Thiès »	0	1	0	0	0
« Saint-Louis »	0	0	0	0	1

Une seule colonne « chaude » (*hot*) par ligne — d'où le nom.

Exercice — encoder à la main

Énoncé

Une colonne « type d'admission » a 4 modalités : urgences, consultation, transfert, programmée. **(a)** Combien de colonnes produit le one-hot ? **(b)** Encoder « transfert ». **(c)** Pourquoi ne pas coder simplement 1, 2, 3, 4 ?

Correction

(a) 4 colonnes binaires. **(b)** (0, 0, 1, 0). **(c)** Les entiers imposeraient un ordre et des écarts inventés — « programmée » (4) serait *trois fois plus loin* des urgences (1) que « consultation » (2), une géométrie qui ne correspond à rien de médical et que les distances de la section suivante prendraient au pied de la lettre.

Mettre à l'échelle : la standardisation

Le problème des échelles

Le constat sur Pima

L'âge court de 21 à 81 ; l'insuline de 14 à 846. Les deux colonnes parlent des unités incomparables — et tout calcul qui les *mélange* (une distance, une somme pondérée) sera dominé par la colonne aux grands nombres, indépendamment de son intérêt médical.

Analogie

Comparer des prix en francs CFA et en euros sans convertir : celui en francs « pèse » des centaines de fois plus dans toute addition — non parce qu'il est plus cher, mais parce que son unité est plus petite. La standardisation est cette conversion : tout le monde dans la même monnaie.

Le z-score, formellement

Définition

Standardiser une colonne, c'est la recentrer sur 0 et la réduire à un écart-type de 1 : chaque valeur devient son **z-score** — sa distance à la moyenne, *comptée en écarts-types*.

$$z = \frac{x - \mu}{\sigma}$$

μ et σ sont la moyenne et l'écart-type de la colonne, **appris sur le train**. Lecture : $z = 0$, valeur parfaitement moyenne ; $z = +2$, deux écarts-types au-dessus — grand, quelle que soit l'unité d'origine. Toutes les colonnes parlent désormais la même langue.

Exemple — un z-score en chiffres

La glycémie sur le train de Pima

Sur les 614 patientes d'entraînement : $\mu_{\text{glycémie}} = 121,6$ et $\sigma_{\text{glycémie}} = 30,0$.

Une patiente à glycémie 180 : $z = \frac{180 - 121,6}{30,0} = \frac{58,4}{30,0} = 1,95$ — presque deux écarts-types au-dessus de la moyenne : nettement élevée.

Même opération pour chaque colonne avec ses μ et σ : l'insuline ($\mu = 139,4$, $\sigma = 84,4$) et l'âge ($\mu = 33,6$, $\sigma = 11,9$) deviennent comparables terme à terme.

Exercice — standardiser à la main

Énoncé

Sur le train, l'âge a pour moyenne $\mu = 33,6$ et pour écart-type $\sigma = 11,9$. Calculer le z-score d'une patiente de 57 ans, et interpréter.

Correction

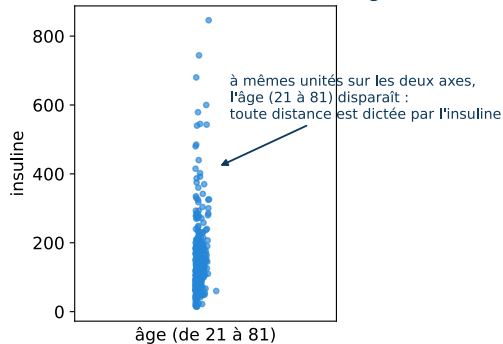
$$z = \frac{57 - 33,6}{11,9} = \frac{23,4}{11,9} = 1,97 \text{ — environ deux écarts-types au-dessus de la moyenne :}$$

cette patiente est nettement plus âgée que la population du registre, exactement comme la glycémie 180 l'était pour sa colonne. Les deux z se comparent directement : c'est tout l'intérêt.

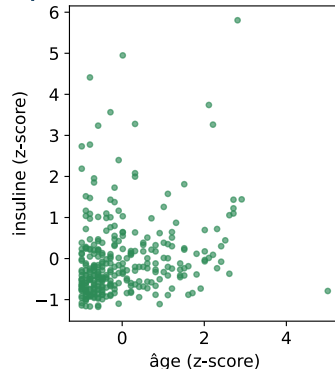
Avant / après, en image

À l'échelle réelle (mêmes unités sur les deux axes), avant / après

Échelles brutes : l'insuline écrase l'âge



Après standardisation : à armes égales



À gauche, âge et insuline tracés *dans les mêmes unités* : l'âge disparaît, toute distance entre patientes est dictée par l'insuline. À droite, après standardisation : les deux variables pèsent selon leurs écarts réels. L'arbre de la Séance 3, qui coupe colonne par colonne, n'y était pas sensible — le modèle qui arrive l'est *viscéralement*.

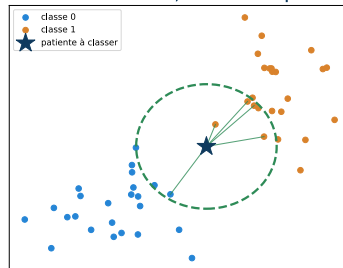
Un second modèle : les k plus proches voisins

L'idée : on prédit comme ses semblables

Analogie

Pour estimer un loyer, un courtier ne résout pas d'équation : il regarde les maisons *voisines et comparables*, et annonce un loyer semblable. Les k plus proches voisins automatisent ce réflexe : « les patientes au profil proche ont connu telle issue ».

$k = 5$ voisins : 4 de classe 1, 1 de classe 0 → prédiction 1



Classer la patiente $\star$: trouver ses $k = 5$ voisines, prendre la classe **majoritaire** — ici 4 contre 1, prédiction 1.

La distance euclidienne, formellement

Mesurer la proximité

« Proche » exige une **distance**. La distance euclidienne entre deux patientes est la longueur du segment qui les relie dans l'espace des caractéristiques — le théorème de Pythagore, en dimension d .

$$d(x, x') = \sqrt{\sum_{j=1}^d (x_j - x'_j)^2}$$

Exemple numérique

En dimension 2 : $d((3, 4), (0, 0)) = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$. Chaque caractéristique contribue par son écart *au carré* — un grand écart sur une seule colonne suffit à dominer toute la somme. Retenir ce détail : il explique tout ce qui suit.

Le modèle kNN, formellement

Définition

Le modèle des k **plus proches voisins** prédit, pour une entrée x , la classe **majoritaire** parmi les k exemples d'entraînement les plus proches de x .

$$f(x) = \text{majorité} \{ y_i : x_i \in V_k(x) \}$$

$V_k(x)$ est l'ensemble des k voisins les plus proches de x dans le jeu d'entraînement. Particularité frappante : il n'y a *rien à entraîner* — **fit** se contente de **mémoriser** les exemples ; tout le travail (calculer les distances, voter) a lieu au moment de **predict**. C'est l'exact opposé de l'arbre, qui travaille à l'entraînement et prédit en un éclair.

Exemple — classer à la main avec $k = 3$

Cinq voisines connues, une patiente à classer en x égal (4 ; 4)

voisine	coord.	classe	distance à (4, 4)	rang
A	(5, 5)	1	$\sqrt{1+1} = 1,41$	1 ^{er}
B	(2, 4)	0	$\sqrt{4+0} = 2,00$	2 ^e
C	(6, 3)	1	$\sqrt{4+1} = 2,24$	3 ^e
D	(1, 1)	0	$\sqrt{9+9} = 4,24$	—
E	(7, 7)	1	$\sqrt{9+9} = 4,24$	—

$V_3 = \{A, B, C\}$, classes $\{1, 0, 1\}$: majorité 1 — $f((4, 4)) = 1$.

Exercice — refaire le vote avec $k = 5$

Énoncé

Mêmes cinq voisines, même patiente (4, 4). Donner la prédiction pour $k = 5$, puis expliquer pourquoi on choisit presque toujours k impair en classification binaire.

Correction

V_5 = toutes les voisines, classes $\{1, 0, 1, 0, 1\}$: trois voix contre deux, prédiction **1**. Un k pair autoriserait des votes à égalité (2-2), qu'il faudrait trancher arbitrairement ; un k impair garantit toujours une majorité nette.

Pourquoi l'échelle est une condition de validité

Trois patientes en deux mesures — âge et insuline

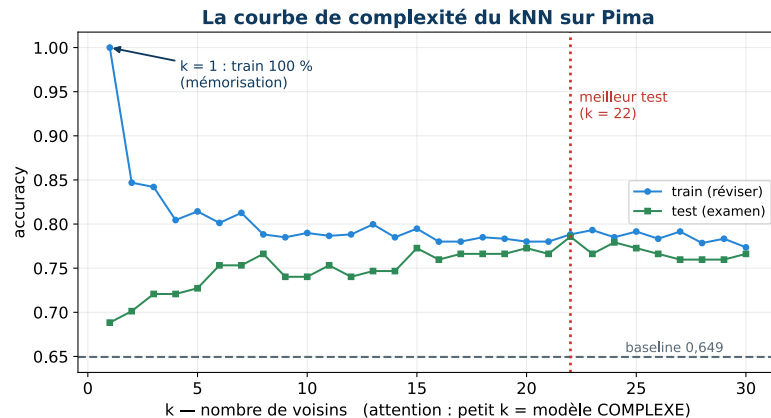
$A = (25, 100)$, $B = (26, 400)$, $C = (60, 105)$. Qui est la plus proche de A ?

En brut : $d(A, B) = \sqrt{1^2 + 300^2} = 300,0$; ; $d(A, C) = \sqrt{35^2 + 5^2} = 35,4$ — la sexagénaire C paraît 8 fois plus proche de la jeune A que B , son quasi-jumeau d'âge : **l'insuline a tout écrasé.**

Standardisé (avec les μ, σ du train, après imputation) : $d(A, B) = 3,55$; ; $d(A, C) = 2,95$ — l'ordre s'inverse, B redevient la plus proche, chaque variable pesant selon ses écarts réels.

Sur Pima en entier, le verdict chiffré : kNN ($k = 5$) **sans** standardisation = 0,714 ; **avec** = 0,727. Pour un modèle à distances, la mise à l'échelle n'est pas un raffinement — c'est une **condition de validité** du calcul lui-même.

Le réglage k : la complexité, encore elle



Attention au sens de lecture : **petit k = modèle complexe**. À $k = 1$, chaque patiente du train est sa propre voisine — train 100 %, test 0,688 : pure mémorisation, le tailleur qui recoud à chaque essayage (**variance**). À k très grand, le vote noie tout détail local — le patron unique (**biais**). Le meilleur test : $k = 22$, accuracy 0,786.

Exercice — diagnostiquer par la paire de scores

Énoncé

Trois réglages du kNN sur Pima : **A** $k = 1 : (1,000; 0,688)$; **B** $k = 22 : (0,79; 0,786)$; **C** $k = 300$: scores faibles et proches tous les deux. Diagnostiquer chacun avec le vocabulaire de la Séance 3.

Correction

A : écart majeur train/test — **sur-apprentissage** (variance) ; mémoriser ses voisines n'est pas généraliser. **B** : scores corrects et proches — le bon compromis. **C** : avec 300 voisines sur 614, le vote tend vers la classe majoritaire globale — **sous-apprentissage** (biais), le modèle redevient presque la baseline. La grille de lecture de la Séance 3 s'applique mot pour mot : seul le bouton de complexité a changé de nom.

La fuite de données

L'idée : un examen dont les questions ont fuité

Analogie

Un étudiant obtient 20/20 — puis on découvre que le sujet de l'examen avait circulé la veille. Sa note ne mesure plus sa compréhension : elle mesure la fuite. En ML, le « sujet » est le jeu de test : toute information qui s'en échappe vers l'apprentissage fabrique un score menteur.

Définition

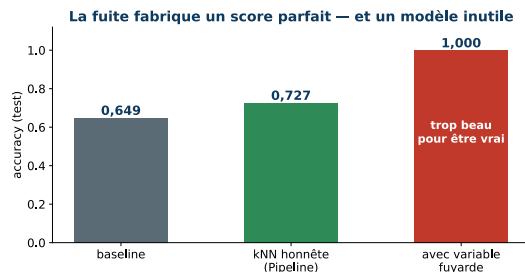
Il y a **fuite de données** (*data leakage*) lorsque l'apprentissage utilise une information qui ne sera **pas disponible au moment de la prédiction** — ou qui provient du jeu de test. Le score mesuré est alors trop beau, et le modèle déployé déçoit.

Deux familles : la **variable fuyarde** (une colonne qui contient la réponse) et le **prétraitement fuyard** (des paramètres appris sur le test). Démonstration de chacune.

Fuite n°1 — la variable fuyarde

La démonstration sur Pima

Ajoutons une colonne « **traitement** » (1 si un traitement antidiabétique a été prescrit).
L'arbre, réentraîné : accuracy = 1,000. Parfait ? Non — le traitement vient *après* le diagnostic : la colonne **contient la réponse**, et à l'admission, au moment de prédire, elle n'existera pas.



Le test à appliquer à *chaque* colonne : « cette information existera-t-elle **au moment de prédire** ? » Si non, elle sort — quel que soit le score qu'elle promet.

Fuite n°2 — le prétraitement fuyard

Le mécanisme

Médianes, μ , σ : tous les paramètres de préparation sont *appris*. Les apprendre sur le jeu **complet** — test inclus — c'est laisser le test influencer l'apprentissage : le modèle a « vu » la moyenne de l'examen avant de le passer.

Sur Pima — un poison discret

Protocole fuyard (standardisation apprise sur les 768 patientes, split ensuite) : 0,727.
Protocole honnête : 0,727 aussi — ici, l'écart est *invisible*. C'est précisément ce qui rend cette fuite dangereuse : elle ne se voit pas au score, le protocole ment *par construction*, et sur d'autres jeux (séries temporelles, petits échantillons) l'écart devient majeur. La règle ne se négocie pas au cas par cas : **tout fit se fait sur le train**.

Exercice — repérer la fuite

Énoncé

Pour chaque protocole, dire s'il y a fuite et laquelle : **(a)** prédire une réadmission avec la colonne « durée totale du séjour » ; **(b)** standardiser tout le jeu, puis faire le split train/test ; **(c)** imputer le train avec les médianes du train, puis le test avec *ces mêmes* médianes.

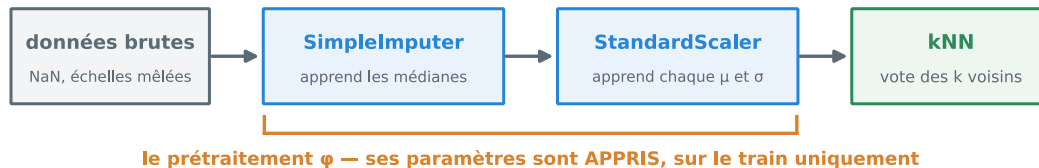
Correction

(a) Variable fuyarde : la durée n'est connue qu'à *la sortie* — au moment de prédire la réadmission, elle n'existe pas encore sous sa forme finale. **(b)** Prétraitement fuyard : μ et σ ont vu le test. **(c) Aucune fuite** — c'est exactement le bon protocole : paramètres appris sur le train, appliqués tels quels au test. Appliquer n'est pas apprendre.

Le Pipeline : souder la chaîne

L'idée : une seule chaîne, soudée

Pipeline scikit-learn : une seule chaîne soudée, $f = m \circ \varphi$



Imputer, standardiser, prédire : trois maillons, chacun avec ses paramètres appris. Les enchaîner à la main multiplie les occasions de fuite (un `fit` de trop, un ordre inversé). Le **Pipeline** soude la chaîne : un seul `fit` (sur le train), un seul `predict` — la fuite de prétraitement devient *impossible par construction*.

Le Pipeline, formellement

Définition

Le modèle complet est la **composée** d'un prétraitement φ et d'un prédicteur m : d'abord transformer, ensuite prédire. Le prétraitement porte ses propres paramètres appris θ_φ — les médianes, les μ et les σ .

$$f(x) = m(\varphi(x)) \quad \text{avec } \theta_\varphi \text{ appris sur le train uniquement}$$

Lire de droite à gauche, comme toute composée : x entre, φ le répare et le met à l'échelle, m prédit. Le **fit** du Pipeline apprend *tout* d'un coup — θ_φ puis les paramètres de m — sur les mêmes données d'entraînement ; le test ne traverse la chaîne qu'en **transform** et **predict**.

Exemple — $\varphi(x)$ pas à pas sur une patiente

La patiente n°525 du test traverse la chaîne

étape	glycémie	insuline	âge
brut	87	NaN	21
après imputation	87	125,5	21
après z-score	$\frac{87 - 121,6}{30,0} = -1,16$	$\frac{125,5 - 139,4}{84,4} = -0,16$	$\frac{21 - 33,6}{11,9} = -1,06$

Chaque nombre utilisé (125,5 ; 121,6 ; 30,0 ; ...) vient du **train** : la patiente, elle, est du test — elle subit la chaîne, elle ne l'a jamais nourrie. Puis m (le kNN) vote sur ce vecteur réparé : $f(x) = m(\varphi(x))$.

En pratique : Pipeline scikit-learn

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

modele = Pipeline([
    ("imputation", SimpleImputer(strategy="median")),
    ("echelle", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=22)),
])
modele.fit(X_train, y_train)      # UN fit : medianes, mu/sigma, voisins
modele.score(X_test, y_test)     # 0.786
```

Le triptyque `fit` / `predict` / `score` de la Séance 3, inchangé — le Pipeline est un modèle comme les autres. Toute la discipline anti-fuite tient dans ces deux lignes finales.

Le bilan chiffré de la séance

Tous les scores sur le même test — 154 patientes

modèle	accuracy (test)
baseline (classe majoritaire)	0,649
arbre profondeur 4 + imputation	0,662
kNN $k = 5$ sans standardisation	0,714
kNN $k = 5$, Pipeline complet	0,727
kNN $k = 22$, Pipeline complet	0,786
<i>n'importe quoi + variable fuyarde</i>	<i>1{,}000 — mensonge</i>

Deux leçons. Le kNN bien préparé bat nettement l'arbre — **aucun modèle n'est roi partout**. Et le saut de 0,662 à 0,786 ne tient pas à un coup de génie : la standardisation rend la distance du kNN *valide* (0,714 $\rightarrow$ 0,727), le choix de modèle et le réglage de k font le reste — la **méthodologie**, pas la chance.

Exercice — remettre la chaîne dans l'ordre

Énoncé

Remettre dans l'ordre les cinq étapes d'une étude propre : **(1)** `score` sur le test ; **(2)** `train_test_split` ; **(3)** remplacer les zéros impossibles par `NaN` ; **(4)** `fit` du Pipeline sur le train ; **(5)** construire le Pipeline (imputation → échelle → modèle). Où une inversion 2 ↔ 4 créerait-elle une fuite ?

Correction

Ordre : **3** → **2** → **5** → **4** → **1**. (Le remplacement 0 → `NaN` n'apprend rien — il peut précéder le split.) Inverser 2 et 4, c'est `fit` avant le découpage : médianes, μ et σ appris sur les 768 patientes, test compris — la fuite n°2 exactement. Le split vient toujours *avant* tout `fit`.

Synthèse

Le protocole complet, version 2

De bout en bout — mis à jour

1. Formuler : (X, y) . ; **2. Auditer** : zéros impossibles → NaN, taux de manquants, échelles, colonnes fuyardes. ; **3.** Découper : `train_test_split`. ; **4.** Construire le **Pipeline** : imputation → encodage → standardisation → modèle. ; **5.** `fit` sur le train — un seul, celui du Pipeline. ; **6.** `score` sur le test, contre la baseline. ; **7.** Diagnostiquer : paire (train, test), réglage de la complexité (k , profondeur).

L'étape 2 est la nouveauté profonde de la séance : avant cette séance, on modélisait ; désormais, on **audite avant de modéliser**.

À retenir — Séance 4

L'essentiel

- Les données réelles cachent leurs trous (zéros impossibles) ; on les expose (**NaN**, taux m_j) puis on les répare par la **médiane** — robuste aux valeurs extrêmes — apprise sur le train.
- Les catégories s'encodent en **one-hot** ; les échelles s'alignent par le **z-score** $z = (x - \mu)/\sigma$.
- Le **kNN** prédit par vote des k voisins : sans mise à l'échelle, sa distance est dominée par les grandes colonnes — l'échelle est une *condition de validité*. Petit k = variance, grand k = biais.
- La **fuite de données** fabrique des scores menteurs : variable fuyarde (la réponse déguisée) ou prétraitement fuyard (**fit** hors du train).
- Le **Pipeline** soude $f = m \circ \varphi$: un seul **fit**, sur le train — la fuite de prétraitement devient impossible par construction.

Pièges fréquents

- Prendre les zéros impossibles pour des mesures — ou jeter la moitié du jeu pour s'en débarrasser.
- Imputer par la moyenne sur des colonnes à valeurs extrêmes (179,4 contre 105 : la moyenne ne ressemble à personne).
- Encoder des catégories par des entiers ordonnés : une géométrie inventée que les distances prennent au sérieux.
- Nourrir un kNN de colonnes non standardisées : la distance n'écoute plus que les grands nombres.
- Croire un score parfait : 1,000 n'est pas une victoire, c'est une **alarme** — chercher la colonne fuyarde.
- Faire un `fit` (imputeur, scaler) avant le split, ou un `fit_transform` sur le test : tout `fit` vit sur le train.

Prochaine séance

Séance 5 — Prédire un nombre : la régression

Jusqu'ici, la cible était une classe. La Séance 5 prédit une **quantité** — une durée de séjour en jours — avec la régression linéaire : ses coefficients lisibles, ses métriques propres (MAE, RMSE, R^2), et le même Pipeline, devenu indispensable : la régression linéaire *refuse* les **NaN**.

Références

- A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*, 3^e éd., O'Reilly, 2022.
- A. C. Müller, S. Guido, *Introduction to Machine Learning with Python*, O'Reilly, 2016.
- G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning*, 2^e éd., Springer, 2021.
- J. W. Smith *et al.*, « Using the ADAP learning algorithm to forecast the onset of diabetes mellitus », 1988 — le jeu Pima.
- Cours : Stanford CS229 ; Cornell CS4780.

Ressources de la séance

- **Notebook** · [TP 4 — Des données propres](#) (32 Ko)
- **TD** · [Fiche d'exercices 4](#) (205 Ko)