

Licence Mathématiques & Applications

Rapport de stage : 2017/2018

---

# L'apprentissage profond sur MIMIC-III : Prédiction de la mortalité sous 24 h

---

Auteur :  
Ayoub ABRAICH

Encadrant :  
Pr. Agathe GUILLOUX

29 mai 2018



# Remerciements



# Table des matières

<b>Introduction</b>	<b>1</b>
<b>1 Exploration et analyse des données</b>	<b>3</b>
1.1 Données MIMIC-III . . . . .	3
1.1.1 Description de MIMIC-III . . . . .	3
1.1.2 Création de MIMIC-III dans une base locale Postgres . . . . .	3
1.2 Définition du problème de l'apprentissage . . . . .	3
1.3 Extraction des données : définition de la cohorte . . . . .	4
1.4 Données longitudinales . . . . .	7
1.5 Aberrations et Corrélation . . . . .	8
1.6 Données manquantes . . . . .	10
1.7 Imputation et transformation des données . . . . .	11
1.8 Division des données . . . . .	12
<b>2 Machine learning</b>	<b>15</b>
2.1 Métriques . . . . .	15
2.2 Algorithmes . . . . .	16
2.2.1 K plus proches voisins : KNN . . . . .	16
2.2.2 Dynamic Time Warping : DTW . . . . .	16
2.2.3 Régression logistique . . . . .	20
2.2.4 Arbre de classification . . . . .	20
2.2.5 Gradient boosting . . . . .	21
2.2.6 Multilayer perceptron : MLP . . . . .	22
2.2.7 Benchmark . . . . .	22

2.3	Implémentation des algorithmes . . . . .	23
<b>3</b>	<b>Résultats</b>	<b>25</b>
3.1	Évaluation et validation du modèle . . . . .	25
3.2	Justification . . . . .	25
<b>4</b>	<b>Conclusion</b>	<b>27</b>
<b>A</b>	<b>Description des Tables utilisées</b>	<b>i</b>
<b>B</b>	<b>Codes : KNN &amp; DTW</b>	<b>v</b>
<b>C</b>	<b>Data-Extraction</b>	<b>xi</b>
<b>D</b>	<b>Data-Wrangling</b>	<b>xxxi</b>
<b>E</b>	<b>Data-Analyse</b>	<b>li</b>

# Introduction

Les unités de soins intensifs (ICU) offrent un soutien aux patients les plus gravement malades dans un hôpital et des traitements radicaux de sauvetage. Les patients sont étroitement surveillés au sein de l'unité de soins intensifs pour aider à la détection précoce et la correction de la détérioration avant qu'elle ne devienne fatale, une approche a été démontrée pour améliorer les résultats [RL Kane ]. Quantifier la santé des patients et prédire les résultats futurs est un domaine important de la recherche en soins intensifs. L'un des résultats les plus pertinents pour l'unité de soins intensifs est la mortalité des patients, qui a mené de nombreuses études vers l'élaboration de modèles de prévision de la mortalité. Généralement, les chercheurs cherchent à améliorer les mesures de performance publiées précédemment, telles que la sensibilité et la spécificité, mais d'autres objectifs peuvent inclure une meilleure interprétation du modèle et une nouvelle extraction de caractéristiques.

Les progrès récents dans l'apprentissage automatique et le réseautage hospitalier ont facilité de meilleurs modèles de prédiction en utilisant des données granulaires plus détaillées. Cependant, interpréter des études qui rapportent des progrès dans la performance de la prédiction de la mortalité est souvent un défi, parce que la comparaison à l'identique est empêchée par le degré élevé d'hétérogénéité entre les études. Par exemple, les approches peuvent différer dans des domaines tels que les critères d'exclusion, le nettoyage des données, la création d'ensembles de formation et de test, etc., ce qui ne permet pas de déterminer clairement où les améliorations de performances ont été obtenues.

Dans de nombreux domaines de l'apprentissage automatique, des ensembles de données tels que ImageNet ont facilité l'analyse comparative et la comparaison entre les études. La clé de ces ensembles de données est qu'ils sont accessibles aux chercheurs, ce qui permet de partager du code et des données pour créer des études reproductibles. Les obstacles au partage de données dans les soins de santé ont limité l'accessibilité des données cliniques hautement granulaires et ont largement empêché la publication d'études reproductibles, mais avec des ensembles de données librement disponibles tels que le Medical Information Mart for Intensive Care (MIMIC-III)[III ] sont réalisables . L'utilisation de modèles de prédiction de la mortalité pour évaluer les soins intensifs dans leur ensemble a été couronnée de succès, tant pour identifier les politiques utiles que pour comparer les populations

de patients. Toutefois, afin de concentrer les contributions à l'état de l'art en matière de prédiction de la mortalité, il devrait être clair où la performance est acquise et des gains supplémentaires pourraient être réalisés.[Alistair E. W. Johnson ]

Ce projet décrit la fouille de donnée sur la base MIMIC-III . L'objectif est de prédire le décès à l'hôpital sur la base MIMIC III.

On va suivre dans ce projet le processus KDD qui est :

1. Sélection et extraction d'un ensemble de données de séries chronologiques multivariées à partir d'une base de données de rangées de millions en écrivant des requêtes SQL.
2. Prétraité et nettoyé la série chronologique en un ensemble de données bien rangé en explorant les données, en gérant les données manquantes (taux de données manquantes > 50%) et en supprimant le bruit / les valeurs aberrantes.
3. Développement d'un modèle prédictif permettant de classer les séries chronologiques biomédicales en mettant en œuvre plusieurs algorithmes tels que l'arbre de décision gradient boost et le KNN avec déformation temporelle dynamique.
4. Résultat de 30% d'augmentation du score F1 par rapport à l'indice de notation médical.

Dans cette étude on se base principalement sur les articles [Sanjay Purushotham ], [Wong ],[Sheth ] et les codes sur MIMIC Code Repository[Johnson ] .



# Chapitre 1

## Exploration et analyse des données

### 1.1 Données MIMIC-III

#### 1.1.1 Description de MIMIC-III

MIMIC III est une base de données de soins critiques accessible au public, mise à jour par le Laboratoire de physiologie computationnelle du Massachusetts Institute of Technology (MIT). Cette base de données intègre des données cliniques détaillées et non identifiées de patients admis dans une unité de soins intensifs du Beth Israel Deaconess Medical Center (BIDMC) à Boston, Massachusetts, entre 2001 et 2012. MIMIC-III contient des données associées à 53 423 hospitalisations distinctes pour les patients adultes (âgés de 15 ans ou plus) et 7870 nouveau-nés admis à l’USI au BIDMC. Les données couvrent 38 597 patients adultes distincts avec 49 785 hospitalisations.

#### 1.1.2 Création de MIMIC-III dans une base locale Postgres

Les détails de création sont sur le site officiel : <https://mimic.physionet.org/tutorials/install-mimic-locally-ubuntu/>

Après avoir installé la base de données MIMIC-III, nous calculons le score SAPS-II en utilisant les données de la base de données. Le code permettant de créer la vue matérialisée pour le score SAPS-II à partir de MIMIC-III est disponible dans GitHub.[Johnson ]

### 1.2 Définition du problème de l’apprentissage

L’objectif est de prédire le décès à l’hôpital (défini comme positif lorsque la durée du séjour à l’hôpital est plus long que le temps de survie) sur la base d’un petit sous-ensemble

de données de l'USI :

- les 24 premières heures du premier séjour en unité de soins intensifs
- 15 mesures

Les étapes nécessaires pour atteindre l'objectif sont :

1. Téléchargez les données MIMIC-III
2. Construire une base de données MIMIC-III PostgreSQL locale
3. Définir la cohorte
4. Prétraitement des données
5. Former les classificateurs pour prédire la mort à l'hôpital
6. Comparer les résultats

Comme on veut prédire la mortalité à partir de  $ihd := \begin{cases} 1, & \text{si heure de sortie} < \text{date de decès} \\ 0, & \text{sinon} \end{cases}$

, on a forcément le problème de machine learning est de type Classification .

On résume les étapes suivies pour cette étude dans le schéma suivant :

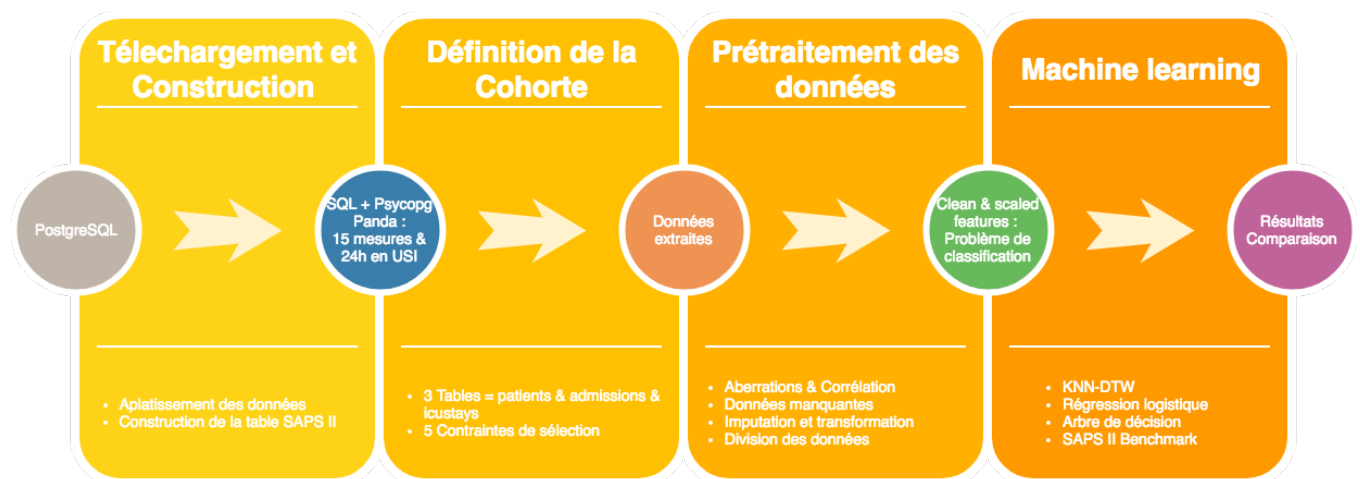


FIGURE 1.1 – Schéma résumant les étapes suivies

### 1.3 Extraction des données : définition de la cohorte

- Description des tables utilisées : Voir Annexe A.1 et Fig 1.2

Bien qu'il y ait 26 tables dans la base de données, nous ne considérons que certaines d'entre elles. Les tableaux relatifs au projet comprennent : admissions, patients, icustays, chartevents, d\_items, outputevents, d\_labitems, labevents, et SAPSII : Voir Table A.1. Tout d'abord, nous définissons la cohorte en fonction de trois tables : patients, admissions et icustays. Les informations contenues dans ces tableaux sont :

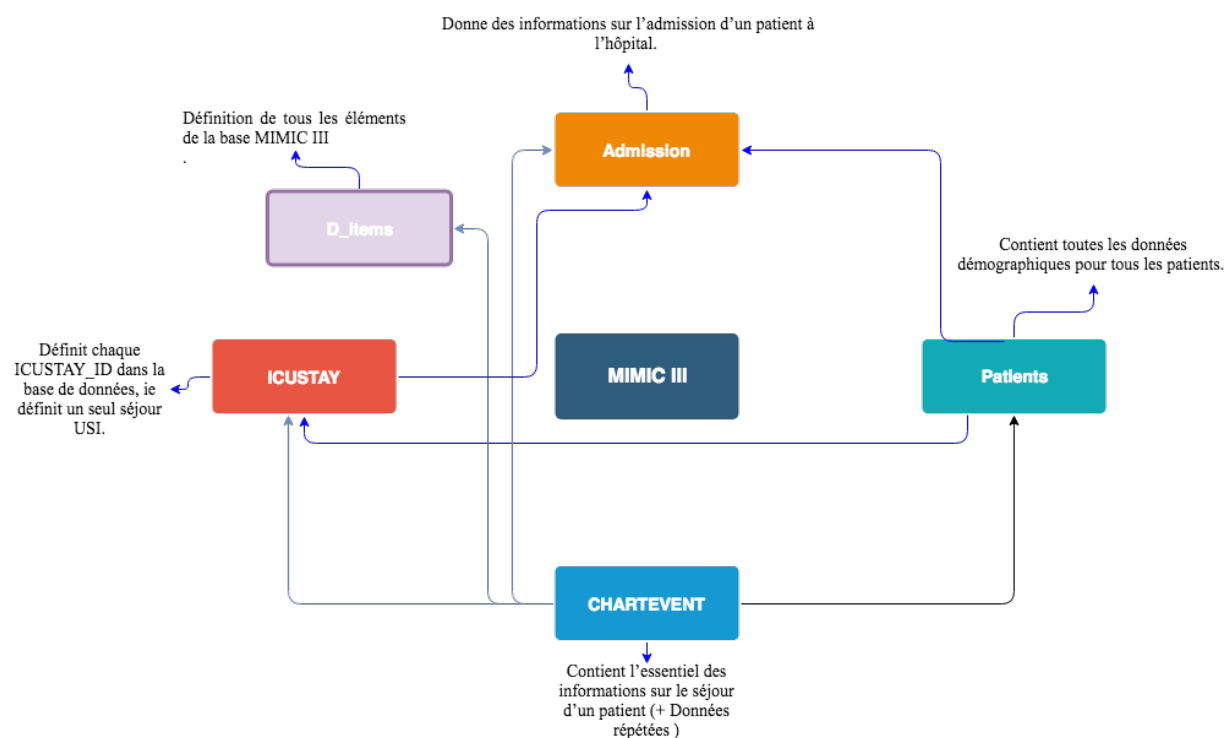


FIGURE 1.2 – Tables utilisées

- patients : informations sur un patient qui ne change pas
- admissions : informations enregistrées à l'admission à l'hôpital
- icustays : informations enregistrées sur l'admission en unité de soins intensifs

Les contraintes utilisées dans la sélection de cohorte sont :

- excl\_age : âge  $\geq 16$  ans
- excl\_lo : la durée du séjour en USI est supérieure à 1 jour mais inférieure à 10 jours pour s'assurer que les données extraites contiennent les 24 premières heures de séjour en USI
- excl\_icustay : ne considérer que la première admission en unité de soins intensifs puisque le patient peut subir plusieurs admissions en unité de soins intensifs dans un hôpital.
- excl\_plan : exclut les événements médicaux planifiés
- excl\_chartevents : exclure les patients sans mesures physiologiques

La distribution d'exclusion sur la sélection des cohortes est illustrée à la figure 1.4.

Condition:	valeur(%)
Total des observations	61532 (100.00%)
excl_age	8109 (13.18%)
excl_lo	19089 (31.02%)

Dimensions de la table de cohorte: (61532, 13)

```

Out[9]:  subject_id  hadm_id  icustay_id   age   los admission_type  \
0      58526    100001    275225  35.48  4.26      EMERGENCY
1      54610    100003    209281  59.91  1.94      EMERGENCY
2       9895    100006    291788  48.92  4.98      EMERGENCY
3      23018    100007    217937  73.82  4.10      EMERGENCY
4        533    100009    253656  60.80  2.49      EMERGENCY

      has_chartevents_data  icustay_order  excl_age  excl_los  excl_icustay  \
0                        1              1         0         0           0
1                        1              1         0         0           0
2                        1              1         0         0           0
3                        1              1         0         0           0
4                        1              1         0         0           0

      excl_plan  excl_chartevents
0             0                0
1             0                0
2             0                0
3             0                0
4             0                0

```

FIGURE 1.3 – Table Cohorte

excl_icustay	15056 (24.47%)
excl_plan	15331 (24.92%)
excl_chartevents	481 (0.78%)
Total exclu	37495 (60.94%)
Total inclu	24037 (39.06%)

Les conditions d'exclusion sur l'âge, la durée du séjour, le premier séjour en unité de soins intensifs et l'événement médical prévu contribuent à la plupart des exclusions. Seule une infime partie des patients est exclue en raison des mesures médicales manquantes. Après la sélection de la cohorte, le nombre d'observations est de 24037, ce qui représente environ 39% du total des observations dans MIMIC-III.

Les données démographiques de la cohorte sont ensuite extraites, dans lesquelles la mort à l'hôpital, *ihd*, est calculée par :

$$ihd := \begin{cases} 1, & \text{si heure de sortie} < \text{date de decès} \\ 0, & \text{sinon} \end{cases}$$

Les données démographiques des patients sont sauvegardées au format CSV et étiquetées «patient\_details.csv».

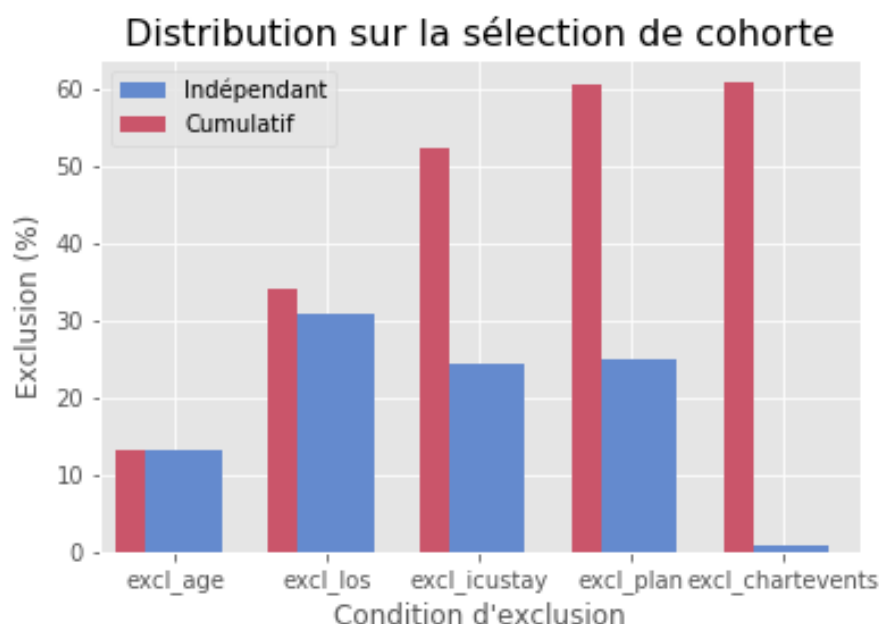


FIGURE 1.4 – La distribution d’exclusion sur la sélection des cohortes

14 séries chronologiques de mesures physiologiques des 24 premières heures de séjour des patients en USI sont extraites. Les mesures comprennent bilirubine, azote uréique du sang (BUN), fraction d’oxygène inspiré (FIO<sub>2</sub>), échelle de coma glasgow (GCS), bicarbonate (HCO<sub>3</sub>), pression partielle d’oxygène (PO<sub>2</sub>), globule blanc (WBC), fréquence cardiaque, potassium, fréquence respiratoire (resp\_rate), sodium, tension artérielle systolique (sys\_BP), température et volume de sortie d’urine (urine\_out).

Ces mesures sont choisies puisqu’elles sont liées au calcul du score SAPS-II. De plus, le score SAPS-II est extrait de la table SAPSII. Enfin, ces données sont stockées séparément dans les fichiers 'chartdata.csv', 'labdata.csv', 'outputdata.csv' et 'sapsii\_score.csv'. Les scripts pour extraire ces fichiers de la base de données MIMIC-III sont disponibles dans le fichier Jupyter notebook 'data\_extraction' voir Annexe C .

## 1.4 Données longitudinales

Les fonctionnalités (Features :Fig.1.5) sont stockées dans les fichiers chartdata, labdata et outputdata CSV. D’abord, nous chargeons les données de ces trois fichiers, puis nous concaténons ces dataframes selon l’axe = 0. En outre, nous convertissons la colonne 'time' du type timedelta en ajoutant la date actuelle. Nous faisons ensuite pivoter la table en faisant passer les colonnes en catégories de telle sorte que chaque rangée montre toutes les variables d’une observation, ce qui est un peu plus près des données ordonnées. (Annexe D)

```
In [43]: feature_df.head()
```

```
Out[43]:
```

		BILIRUBIN	BUN	FI02	GCS	HCO3	PO2	WBC	heart rate	potassium	resp_rate	sodium	sys_BP	temperature	urine_out	age
icustay_id	time															
200003	2018-05-20 21:49:56	3.5	21.0	NaN	NaN	23.0	NaN	14.8	NaN	3.1	NaN	140.0	NaN	NaN	NaN	48.3
	2018-05-21 00:09:56	NaN	NaN	NaN	5.0	NaN	NaN	NaN	119.0	NaN	35.0	NaN	91.0	NaN	NaN	48.3
	2018-05-21 00:24:56	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	230.0	48.3
	2018-05-21 01:09:56	NaN	NaN	NaN	NaN	NaN	NaN	NaN	122.0	NaN	34.0	NaN	81.0	39.0	NaN	48.3
	2018-05-21 01:34:56	NaN	NaN	NaN	NaN	NaN	NaN	NaN	115.0	NaN	33.0	NaN	98.0	NaN	NaN	48.3

FIGURE 1.5 – Features

## 1.5 Aberrations et Corrélation

Nous fusionnons d'abord les données extraites dans la dernière section en deux tables décrivant la fonctionnalité et la cible. 'Chartdata.csv', 'labdata.csv' et 'outputdata.csv' sont fusionnés pour représenter les données d'entité. 'Patient\_details.csv' et 'sapsii\_score.csv' sont fusionnés pour représenter les données cibles 1.6.

	in_hospital_death	sapsii_prob	sapsii_prediction
icustay_id			
200003	0	0.106398	0
200007	0	0.029295	0
200014	0	0.305597	0
200019	1	0.552904	1
200021	0	0.326364	0

FIGURE 1.6 – La tête de la table : données cibles

Les détails de la fusion sont indiqués dans le bloc-notes de Jupyter, 'data\_analyse.ipynb' Annexe E. Les données caractéristiques résultantes sont décrites par une série temporelle multivariée à indices multiples (identifiant et heure du patient) avec 15 caractéristiques différentes, représentant diverses mesures médicales. Les données cibles résultantes sont décrites par un tableau 2D NX3, dans lequel N est le nombre d'observations. Il y a 3 variables dans les données cibles, à savoir le décès à l'hôpital, le score SAPS-II et sa

probabilité. Le nettoyage des données est principalement effectué pour les données de caractéristiques. Nous définissons d'abord les valeurs aberrantes. La plage normale de GCS est comprise entre 1 et 5, de sorte que les observations hors de cette plage sont considérées comme aberrantes. Pour la variable «age», la plage doit être comprise entre 16 et 91,4 (âge maximum) et les valeurs hors de cette plage sont considérées comme aberrantes. Pour le reste des variables, z-score est calculé et est utilisé pour définir les valeurs aberrantes. Le score z est utilisé pour comparer une observation à une variable normale standard et est calculé par :

$$z := \frac{x_{i,j}^k - E_j(x)}{\sigma_j(x)}$$

Où  $x_{i,j}^k$  représente la valeur de la caractéristique  $j$  dans l'observation à l'instant  $i$  pour le patient avec k ID.  $E_j(x)$  est l'attente de  $x_j$  tout le temps et des patients, et  $\sigma_j(x)$  est la dérivation standard de  $x_j$  sur tous les temps et les patients. En expression mathématique :

$$E_j(x) := \frac{\sum_{i,k} x_{i,j}^k}{n}$$

$$\sigma_j(x) := \sqrt{\frac{\sum_{i,k} (x_{i,j}^k - E_j(x))^2}{n - 1}}$$

Si l'observation n'est pas comprise entre 3,5 et -3,5 dérivations standard de la moyenne, ce qui suggère qu'elle est loin de la moyenne et peut être considérée comme une valeur aberrante. Tous les outliers sont considérés comme des données manquantes et d'autres traitements sont nécessaires. La distribution du score z après suppression des valeurs aberrantes est illustrée à la Figure 1.7

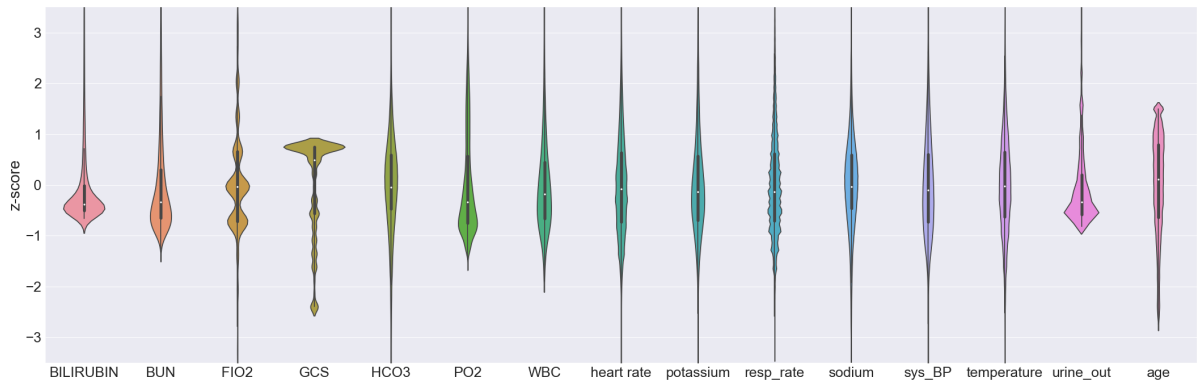


FIGURE 1.7 – La distribution du score z

La matrice de corrélation de la série temporelle multivariée est calculée et la matrice est visualisée à l'aide d'une carte thermique illustrée à la figure 1.8. À partir de la courbe, il n'y a pas de corrélation significative entre les caractéristiques. Par conséquent, nous devrions considérer toutes les caractéristiques dans l'analyse ultérieure.

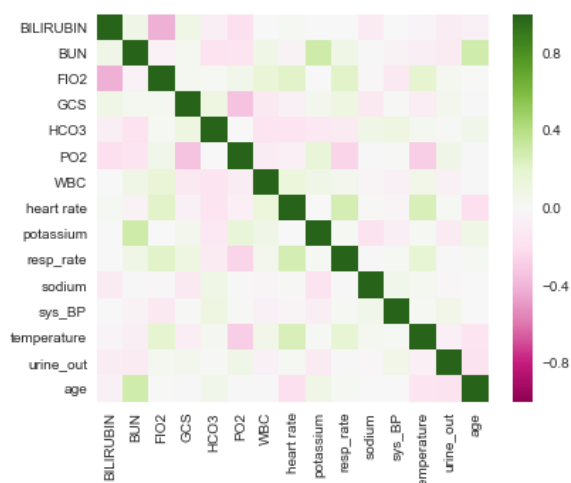


FIGURE 1.8 – La matrice de corrélation de la série temporelle multivariée.

## 1.6 Données manquantes

La série temporelle extraite est espacée de manière inégale, et ainsi la série temporelle est ré-échantillonnée avec une fréquence bimensuelle. Les données obtenues contiennent beaucoup de données manquantes puisque les mesures ne couvrent pas les 24 premières heures du séjour en unité de soins intensifs. Le rapport des données manquantes dans les 24 premières heures est résumé dans la Fig. 1.9 et 1.10. Idéalement, nous voulons une série

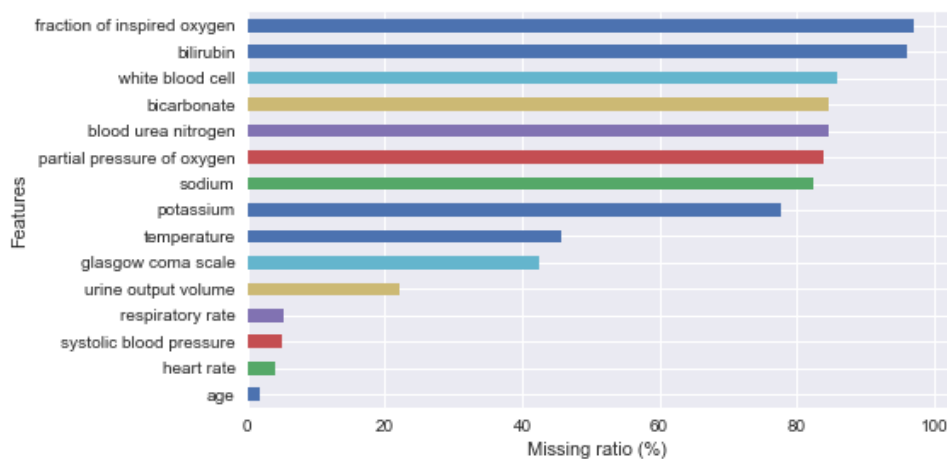


FIGURE 1.9 – Données manquantes

temporelle multivariée complète qui peut capturer l'information temporelle. Malheureusement, la moitié des fonctionnalités ont plus de 75% de données manquantes dans ce cas. Seuls le volume de sortie d'urine, la fréquence respiratoire, la pression artérielle systolique,



```
In [33]: resampled_df.count()

Out[33]: BILIRUBIN      11470
          BUN           44356
          FIO2          8308
          GCS          165978
          HCO3          44229
          PO2           46203
          WBC           40430
          heart rate    276461
          potassium     64205
          resp_rate     273042
          sodium        50998
          sys_BP        273612
          temperature   156495
          urine_out     224168
          age           283368
          dtype: int64
```

FIGURE 1.10 – Bilan : Données manquantes

la fréquence cardiaque et l'âge ont plus de 50% des données.

## 1.7 Imputation et transformation des données

Les données manquantes sont traitées par imputation selon les étapes suivantes :

- appliquer une interpolation linéaire sur la série chronologique multivariée pour chaque patient
- remplir les données manquantes en utilisant la prochaine observation valide pour chaque patient

Certaines observations sont toujours manquantes après ces imputations, car il y a des données manquantes pour certaines fonctions chez certains patients. Dans ce cas, Nous rééchantillons les séries chronologiques irrégulières en séries temporelles régulières avec un intervalle de temps de 2 heures Fig. 1.11.

Ensuite, nous interpolons la série chronologique par chaque patient. Enfin, nous imputons (1.12) l'utilisation de la moyenne si toute la série temporelle est manquante. La fonction 'MinMaxScalar' dans Sklearn est appliquée pour transformer les entités en mettant à l'échelle chaque caractéristique dans la plage de [0, 1].

Deux séries temporelles multivariées pour deux patients imputées et transformées est représentée à titre d'exemple, ce qui est illustré à la Fig. 1.13 et 1.14

icustay_id	time	BILIRUBIN	BUN	FIO2	GCS	HCO3	PO2	WBC	heart rate	potassium	resp_rate	sodium	sys_BP	temperature	urine_out	age
200003	2018-05-25 00:00:00	NaN	NaN	NaN	5.0	NaN	NaN	NaN	118.200000	NaN	32.60	NaN	89.000000	39.00	230.00	48.3
	2018-05-25 02:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	114.400000	NaN	31.20	NaN	85.800000	38.03	90.00	48.3
	2018-05-25 04:00:00	3.4	20.0	NaN	5.0	18.0	NaN	40.2	112.166667	3.2	33.00	141.0	96.333333	36.83	128.50	48.3
	2018-05-25 06:00:00	NaN	NaN	NaN	NaN	NaN	91.0	NaN	109.000000	NaN	33.25	NaN	104.800000	36.44	263.75	48.3
	2018-05-25 08:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	93.000000	NaN	37.00	NaN	112.500000	NaN	300.00	48.3

FIGURE 1.11 – Rééchantillage

icustay_id	time	BILIRUBIN	BUN	FIO2	GCS	HCO3	PO2	WBC	heart rate	potassium	resp_rate	sodium	sys_BP	temperature	urine_out	age
200003	2018-05-25 00:00:00	0.187845	0.180952	0.499414	1.0	0.333333	0.132075	0.805221	0.749606	0.314815	0.854054	0.578947	0.308176	0.850250	0.067565	0.42823
	2018-05-25 02:00:00	0.187845	0.180952	0.499414	1.0	0.333333	0.132075	0.805221	0.719685	0.314815	0.816216	0.578947	0.288050	0.688852	0.026385	0.42823
	2018-05-25 04:00:00	0.187845	0.180952	0.499414	1.0	0.333333	0.132075	0.805221	0.702100	0.314815	0.864865	0.578947	0.354298	0.489185	0.037709	0.42823
	2018-05-25 06:00:00	0.191298	0.179762	0.499414	1.0	0.359848	0.132075	0.814508	0.677165	0.312500	0.871622	0.588816	0.407547	0.424293	0.077492	0.42823
	2018-05-25 08:00:00	0.194751	0.178571	0.499414	1.0	0.386364	0.139365	0.823795	0.551181	0.310185	0.972973	0.598684	0.455975	0.420133	0.088155	0.42823

FIGURE 1.12 – Scaling et Imputation

## 1.8 Division des données

L'ensemble de données est déséquilibré et contient environ 11,4% d'échantillons positifs. Afin de préserver le pourcentage d'échantillons pour chaque classe, une répartition aléatoire stratifiée est appliquée à l'ensemble de données, la taille de test est définie sur 20% de l'ensemble de données d'origine, qui contient 4806 échantillons. D'un autre côté, l'ensemble de données d'apprentissage contient 19220 échantillons. Enfin, les ensembles de données d'entraînement et de test sont divisés en 'X\_train' [1.15](#), 'X\_test' [1.16](#), 'y\_train' et 'y\_test' [1.17](#).

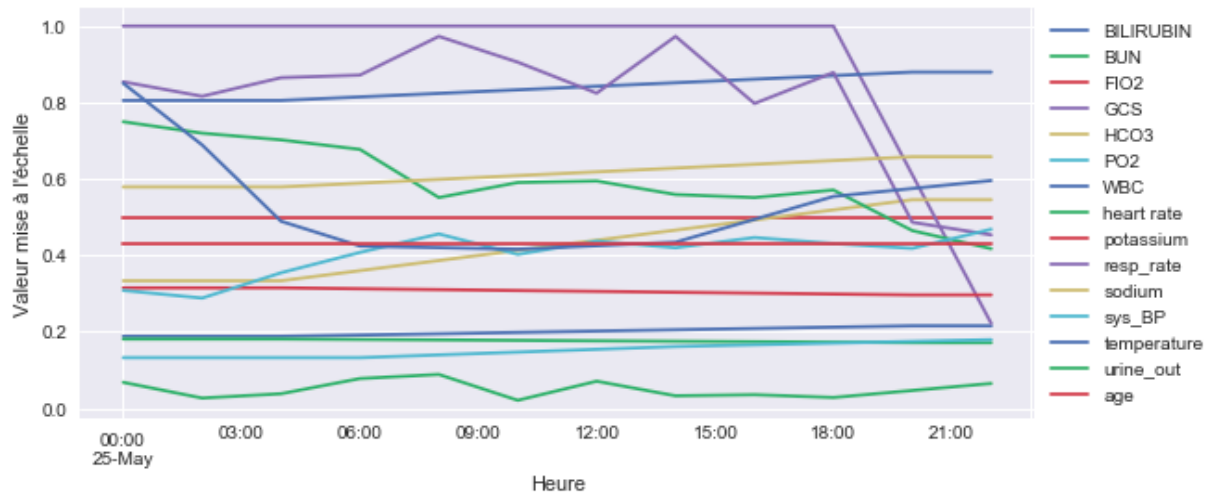


FIGURE 1.13 – Exemple : Patient 1

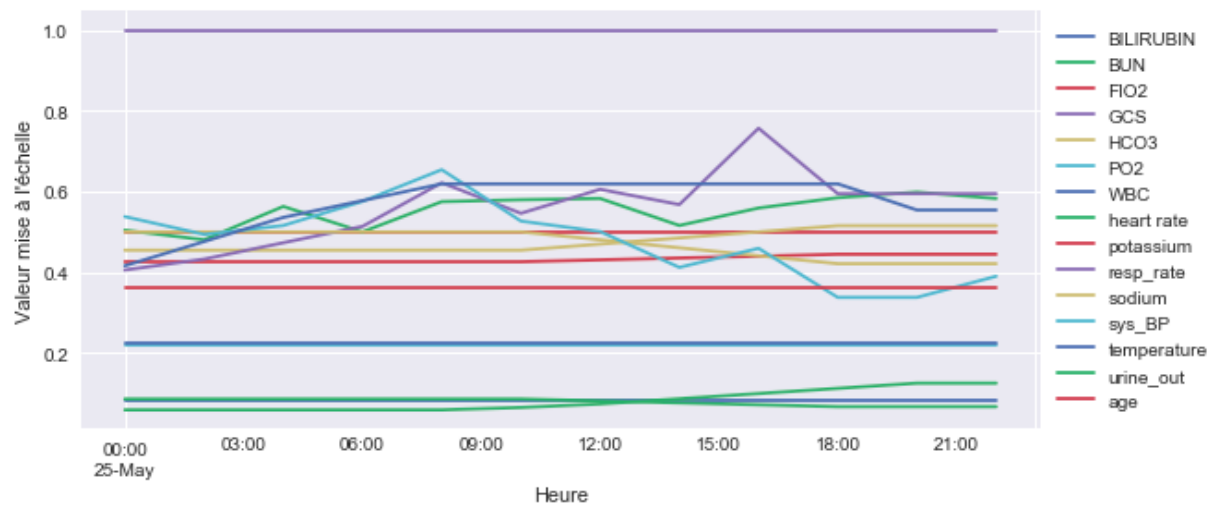


FIGURE 1.14 – Exemple : Patient 2

		BILIRUBIN	BUN	FIO2	GCS	HCO3	PO2	WBC	heart rate	potassium	resp_rate	sodium	sys_BP	temperature	urine_out	age
id	time															
0	0	0.082443	0.171429	0.499414	1.0	0.484848	0.219736	0.068273	0.307087	0.222222	0.513514	0.631579	0.723270	0.564060	0.014619	1.0
	1	0.082443	0.171429	0.499414	1.0	0.484848	0.219736	0.068273	0.232283	0.222222	0.432432	0.631579	0.691824	0.512895	0.014619	1.0
	2	0.082443	0.171429	0.499414	1.0	0.515152	0.219736	0.074297	0.225722	0.305556	0.445946	0.605263	0.603774	0.461730	0.058740	1.0
	3	0.082443	0.171429	0.499414	1.0	0.545455	0.219736	0.080321	0.267717	0.388889	0.500000	0.578947	0.679245	0.397671	0.026385	1.0
	4	0.082443	0.171429	0.499414	1.0	0.575758	0.219736	0.086345	0.275591	0.472222	0.581081	0.552632	0.676101	0.401830	0.073448	1.0

FIGURE 1.15 – Tête de table : X\_train

		BILIRUBIN	BUN	FIO2	GCS	HCO3	PO2	WBC	heart rate	potassium	resp_rate	sodium	sys_BP	temperature	urine_out	age
id	time															
0	0	0.082443	0.171429	0.499414	1.0	0.484848	0.219736	0.068273	0.307087	0.222222	0.513514	0.631579	0.723270	0.564060	0.014619	1.0
	1	0.082443	0.171429	0.499414	1.0	0.484848	0.219736	0.068273	0.232283	0.222222	0.432432	0.631579	0.691824	0.512895	0.014619	1.0
	2	0.082443	0.171429	0.499414	1.0	0.515152	0.219736	0.074297	0.225722	0.305556	0.445946	0.605263	0.603774	0.461730	0.058740	1.0
	3	0.082443	0.171429	0.499414	1.0	0.545455	0.219736	0.080321	0.267717	0.388889	0.500000	0.578947	0.679245	0.397671	0.026385	1.0
	4	0.082443	0.171429	0.499414	1.0	0.575758	0.219736	0.086345	0.275591	0.472222	0.581081	0.552632	0.676101	0.401830	0.073448	1.0

FIGURE 1.16 – Tête de table : X\_test

1	y_train.head()		
	in_hospital_death	sapsii_prob	sapsii_prediction
id			
0	0.0	0.096698	0.0
1	0.0	0.391926	0.0
2	0.0	0.140051	0.0
3	0.0	0.096698	0.0
4	0.0	0.152870	0.0

1	y_test.head()		
	in_hospital_death	sapsii_prob	sapsii_prediction
id			
19220	0.0	0.004584	0.0
19221	0.0	0.079390	0.0
19222	1.0	0.166523	0.0
19223	0.0	0.052195	0.0
19224	0.0	0.305597	0.0

FIGURE 1.17 – Têtes de tables : y\_train et y\_test

# Chapitre 2

## Machine learning

### 2.1 Métriques

En USI, il est mauvais d'avoir beaucoup de faux négatifs puisque nous ne voulons pas manquer de patients à haut risque. Dans ce cas, le rappel doit être pris en compte, car un rappel plus élevé indique un taux de faux négatifs plus faible. Cependant, un classificateur naïf qui prédit toujours positif peut atteindre 100% de rappel, mais il reste un mauvais classificateur puisqu'il ne se classe pas du tout. La lacune du rappel peut être résolue en considérant la précision, qui est définie comme le rapport de la vraie positive à la condition prédite positive. Il y a beaucoup de mappage du rappel et de la précision à un score unique, par exemple, une moyenne simple d'entre eux, peut être utilisé pour représenter la performance du classificateur. Dans notre cas, le score F1 qui est un moyen harmonique de rappel et de précision est choisi car il est robuste aux données déséquilibrées. L'expression mathématique du score F1 est [Wong ] :

$$F1 := \frac{2 \cdot \text{rappel} \cdot \text{precision}}{\text{rappel} + \text{precision}}$$

Le coefficient de corrélation de Matthews (MCC) est une autre métrique qui est robuste sur les données déséquilibrées, et il considère également le vrai négatif. Par conséquent, il est choisi de comparer les performances du classificateur dans ce projet. Le MCC est calculé :

$$MCC := \frac{n_{vp} \cdot n_{vn} - n_{fp} \cdot n_{fn}}{\sqrt{(n_{vp} + n_{fp})(n_{vp} + n_{fn})(n_{vn} + n_{fp})(n_{vn} + n_{fn})}}$$

où  $n_{vp}$  est le nombre de vrais positifs,  $n_{vn}$  est le nombre de vrais négatifs,  $n_{fp}$  est le nombre de faux positifs, et  $n_{fn}$  est le nombre de faux négatifs. De plus, si l'une des quatre sommes du dénominateur est zéro,  $MCC$  est égal à zéro.

'F1\_score' et 'matthews\_corrcoef' sont importés du module 'sklearn.metrics' pour calculer le score F1 et le coefficient de corrélation de Matthews (MCC).

## 2.2 Algorithmes

La méthode de classification des séries temporelles choisie [Z.Xing ] est DTW - Dynamic Time Warping -qui est une des mesures de similarité qui calcule les meilleures correspondances entre deux séries temporelles. Une fois la similarité de distance entre deux séries de temps définie, un algorithme basé sur la distance classique tel que k-NN, SVM peut être appliqué pour classer les séries temporelles. DTW avec kNN est l'une des meilleures solutions connues pour les problèmes de séries chronologiques dans une variété de domaines [Keogh ]. Par conséquent, nous essayons d'abord de classer notre série temporelle médicale multivariée en utilisant DTW avec le classificateur kNN. Les algorithmes DTW et kNN sont implémentés via Python dans 'dtw.py' et 'KNN.py'.(Github) - Voir Annexe [B](#)

Nous présentons les principes des algorithmes utilisées :

### 2.2.1 K plus proches voisins : KNN

C'est méthode très intuitive qui classe les exemples non étiquetés sur la base de leur similarité avec les exemples de la base d'apprentissage [KNN ].

k-NN nécessite seulement :

- Un entier k
- Une base d'apprentissage
- Une métrique pour la proximité

Exemple :

- Dans cet exemple [2.1](#) on a 3 classes et le but est de trouver la classe de l'inconnu x.
- On prend la distance Euclidienne et k=5 voisins.
- Des 5 plus proches voisins, 4 appartiennent à  $\omega_1$  et 1 appartient à  $\omega_3$ , donc x est affecté à  $\omega_1$ , la classe majoritaire.

Formellement [Guilloux ] :

- On considère l'ensemble  $I_x$  composé des k indices de  $\{1, \dots, n\}$  pour lesquels les distances  $\|x - X_i\|$  sont minimales.
- On pose :

$$\hat{c}(x) := \underset{k \in \{1, \dots, K\}}{\operatorname{Argmax}} \# \{i \in I_x, Y_i = k\}$$

- En pratique, il faut choisir la distance et  $K$  .
- La complexité temporelle de KNN est  $O(nmK)$

### 2.2.2 Dynamic Time Warping : DTW

Dans cette section, nous décrivons l'algorithme DTW [Mueen ], qui est utilisé pour mesurer la distance entre deux séries chronologiques. Il a été initialement proposé en 1978

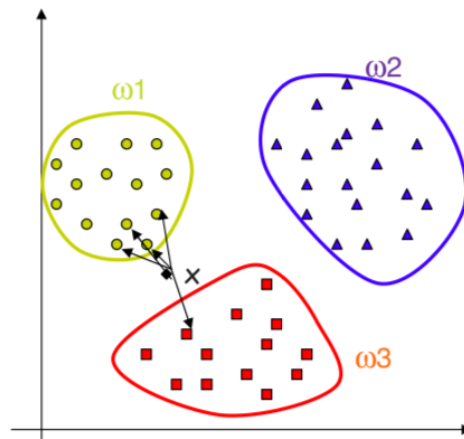


FIGURE 2.1 – Exemple : KNN

par Sakoe et Chiba[] pour la reconnaissance vocale, et il a été utilisé jusqu'à aujourd'hui pour l'analyse de séries temporelles. DTW est l'une des mesures les plus utilisées de la similarité entre deux séries chronologiques, et calcule l'alignement global optimal (2.2) entre deux séries chronologiques, en exploitant les distorsions temporelles entre elles.

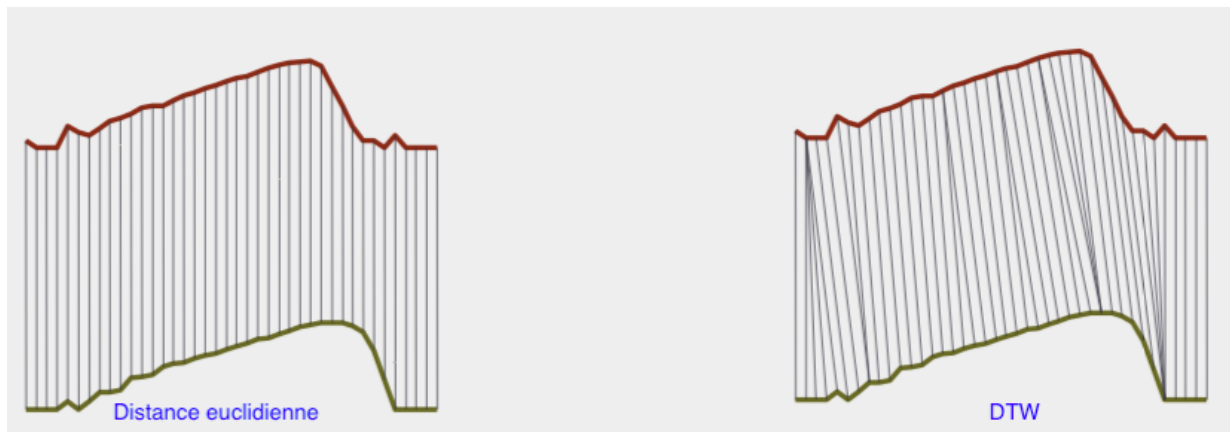


FIGURE 2.2 – Distance euclidienne Vs DTW

**Calcul du DTW :** [Elsworth ] Soient  $X := \{x_i\}_{i=1}^m$  et  $Y := \{y_i\}_{i=1}^n$  deux series temporelles :

1. Construction de la matrice  $D$  de taille  $n \times m$  définie par :  $D_{i,j} := d(x_i, y_j)$  où  $d(x_i, y_i) = |x_i - y_i|$ .
2. Un chemin de déformation  $w$  est un ensemble contigu d'éléments de matrice qui définit un mappage entre  $X$  et  $Y$  qui satisfait aux conditions suivantes :(2.5)
  - (a) Conditions aux limites :  $w_1 = (1, 1)$  et  $w_k = (m, n)$  où  $k$  est la longueur du chemin de déformation.

- (b) Continuité : si  $w_i = (a, b)$  alors  $w_{i-1} = (a', b')$  où  $a - a' \leq 1$  et  $b - b' \leq 1$ .
- (c) Monotonie : si  $w_i = (a, b)$  alors  $w_{i-1} = (a', b')$  où  $a - a' \geq 0$  et  $b - b' \geq 0$ .
3. La distance DTW est donc :  $d_{DTW}(X, Y) = \min \sum_{i=1}^k D(w_i)$
4. En pratique, nous utilisons la programmation dynamique pour empêcher la construction de la matrice entière . On définit  $\gamma(i, j)$  la distance cumulée comme :

$$\gamma(i, j) := d(x_i, y_j) + \min(\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1))$$

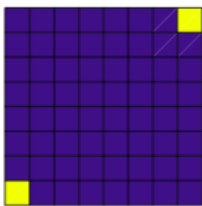
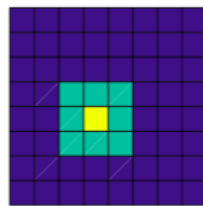
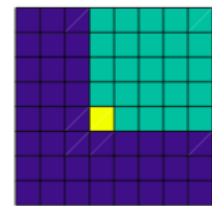
**Boundary****Continuity****Monotonicity**

FIGURE 2.3 – Conditions : Chemain de déformation

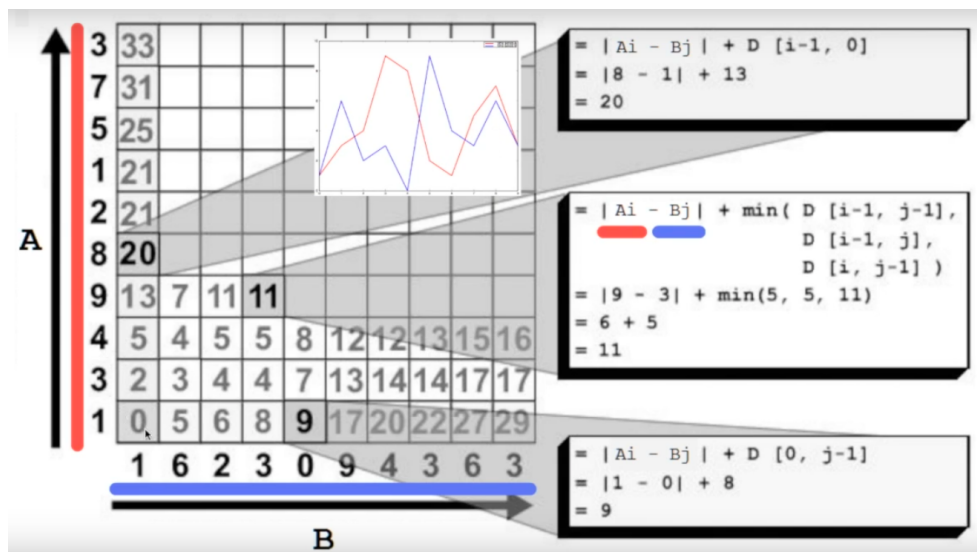


FIGURE 2.4 – Exemple : calcul de la distance cumulée

- La complexité temporelle de DTW est  $O(n^2)$



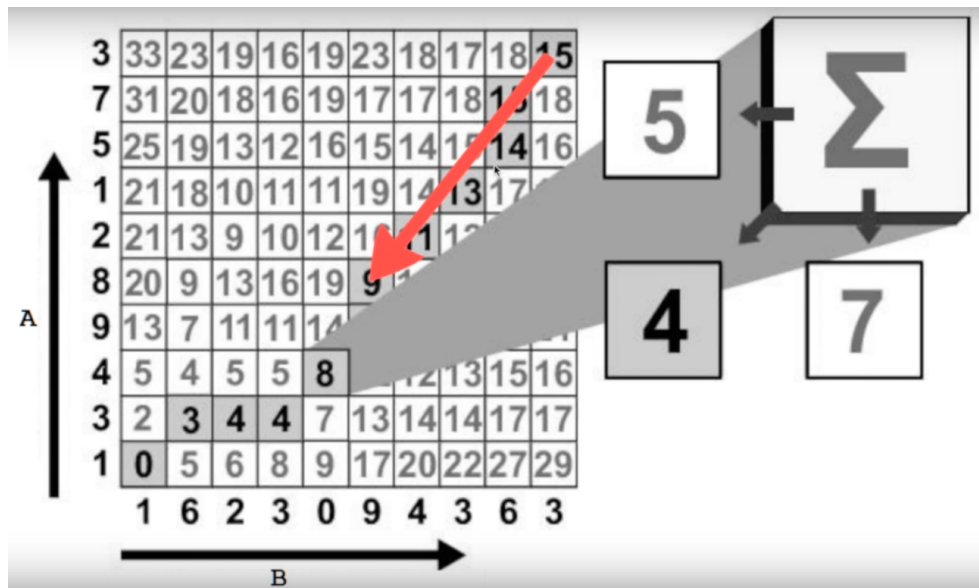


FIGURE 2.5 – Exemple : calcul de la distance cumulée

En plus d'utiliser le DTW kNN, nous utilisons également des classificateurs conventionnels (Logistic regression, Decision Tree Classifier et Keras classifier) pour prédire la mort à l'hôpital. Mais, nous devons d'abord extraire de nouvelles fonctionnalités de la série temporelle multivariée et réduire les dimensions des données de la 3D à la 2D.

Nous formons d'abord le modèle avec un sous-ensemble de données d'entraînement et de test pour une meilleure gestion du temps. 1922 (10%) et 480 (10%) observations d'entraînement et de test sont utilisées. Le résultat du classificateur DTW 3-NN est résumé dans le Tableau 2.2.2

	Temps de formation (s)	Temps de prédiction (s)	Score F1	Score MCC
DTW kNN	0.003	2244	0.2749	0.2612
Algorithme naïf	NA	NA	0.22	0.0

Le classificateur k-NN, l'apprenant paresseux, ne forme pas un modèle et son temps d'entraînement est négligeable. Cependant, il a fallu environ 2244 secondes pour prédire l'étiquette du sous-ensemble de données de test, ce qui suggère qu'il est coûteux en calcul. La performance de la prédiction DTW k-NN n'est pas prometteuse car le classifieur naïf qui prédit toujours positif peut atteindre un score F1 comparable à celui-ci. Bien que l'utilisation de l'ensemble de données d'apprentissage complet puisse améliorer les performances de DTW kNN, le coût de calcul est trop élevé. De plus, le taux manquant de la série temporelle multivariée est élevé comme le montre la figure 1.9, qui indique que certaines variables peuvent contenir moins d'informations temporelles. La perte d'information temporelle a un impact négatif sur le classificateur basé sur DTW, ce qui est démontré dans

l'article [Mikalsen ]. La deuxième tentative de ce projet consiste à extraire de nouvelles fonctionnalités de la série chronologique multivariée. Cela supprime la dépendance temporelle des données et réduit les données d'apprentissage de 3 dimensions à 2 dimensions de sorte que les méthodes de classification habituelles telles que la régression logistique sont applicables.

Dans ce projet, la régression logistique, l'arbre de décision, le boost de gradient, et les classificateurs de perceptron multicouche sont utilisés.

### 2.2.3 Régression logistique

La régression logistique est le plus utilisé des algorithmes de classification [Guilloux ]. Elle a un temps d'apprentissage rapide et est robuste aux données bruitées. Par conséquent, il est utilisé comme classificateur de base. En régression logistique, nous formons une fonction de la forme,

$$P(y = 1 | x) = \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$$

pour minimiser la fonction de coût :

$$J(\theta) := - \sum_i (y_i \log(P(y = 1 | x_i)) + (1 - y_i) \log(1 - P(y = 1 | x_i)))$$

C'est un problème de minimisation convexe et régulier, il y a de nombreux algorithmes (descente de gradient, Newton, etc).  $\underset{\theta}{\operatorname{Argmin}} J(\theta)$  renvoie  $\theta$ , qui est utilisé pour calculer la probabilité d'être positif ou négatif basé sur  $x$ .

### 2.2.4 Arbre de classification

L'arbre de classification est un apprenant basé sur la logique, qui partitionne récursivement l'espace échantillon de sorte que les échantillons de la même classe sont regroupés. [Guilloux ]

- Construction d'un arbre de classification :

Approche "top-bottom" (2.6) :

- On commence par une région qui contient toutes les données
- On coupe récursivement les régions par rapport à une variable et une valeur de cette variable

- Critère de segmentation :

Usuellement, les algorithmes pour construire les arbres de décision sont construits en divisant l'arbre du sommet vers les feuilles en choisissant à chaque étape une variable d'entrée qui réalise le meilleur partage de l'ensemble d'objets, comme décrit précédemment<sup>4</sup>. Pour choisir la variable de séparation sur un nœud, les algorithmes testent les différentes variables d'entrée possibles et sélectionnent celle qui maximise un critère donné.

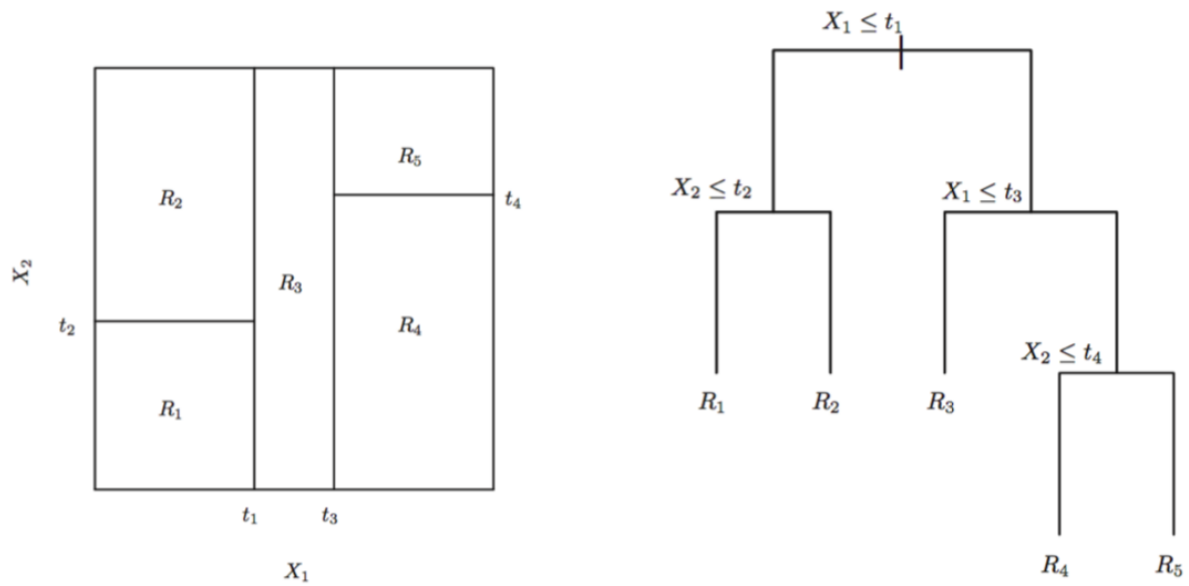


FIGURE 2.6 – Construction d'un arbre de classification

Dans le cas des arbres de classification, il s'agit d'un problème de classification automatique. Le critère d'évaluation des partitions caractérise l'homogénéité (ou le gain en homogénéité) des sous-ensembles obtenus par division de l'ensemble. Ces métriques sont appliquées à chaque sous-ensemble candidat et les résultats sont combinés (par exemple, moyennés) pour produire une mesure de la qualité de la séparation.

Il existe un grand nombre de critères de ce type, les plus utilisés sont l'entropie de Shannon, l'indice de diversité de Gini et leurs variantes.[Wikipedia ]

Dans Sklearn , l'apprenant par défaut est formé pour réduire l'impureté moyenne de Gini afin d'obtenir un gain d'information par itération. L'impureté moyenne de Gini est :

$$I_G(S, A) := \sum_i \frac{|S_i|}{|S|} I_G(S_i)$$

où  $S_i$  est la taille de la partition,  $I_G(S_i)$  est l'impureté de Gini de la partition  $S_i$  :

$$I_G(S_i) := 1 - \sum_j p_j^2$$

où  $p_j$  représente la proportion d'exemples de la classe  $j$  sur  $S_i$ .

### 2.2.5 Gradient boosting

Gradient boosting est une technique d'apprentissage automatique pour les problèmes de régression et de classification, qui produit un modèle de prédiction sous la forme d'un

ensemble de modèles de prédiction faibles, typiquement des arbres de décision. Il construit le modèle par étapes comme les autres méthodes de boosting, et il les généralise en permettant l'optimisation d'une fonction de perte différentiable arbitraire. [Wikipedia ]

L'ensemble d'apprenants mis à jour est : [Wong ]

$$F_{i,m+1}(x) := F_{i,m}(x) + \eta h_i(x)$$

où  $\eta$  est un taux d'apprentissage,  $h_i(x)$  est le nouvel apprenant,  $F_{i,m}$  est le modèle de la classe  $i$  et est utilisé pour calculer la probabilité d'être en classe  $i$  en :

$$P_i := \frac{\exp(F_i)}{\sum_i \exp(F_i)}$$

Le nouvel apprenant peut être calculé en minimisant la KL-divergence (entropie relative) :

$$h_i(x) := y_i(x) - P_i(x)$$

où  $y_i$  est la vraie probabilité (le vrai étiquetage). Les prédictions faites à partir de ces apprenants sont donc pondérées donnant une prédiction globale au problème. De plus, le boosting peut contrôler à la fois le biais et la variance pour améliorer la performance de prédiction. Cependant, il est coûteux en calcul parce que plusieurs modèles sont requis. Puisque de nombreux apprenants sont impliqués, il est difficile d'interpréter la prédiction faite.

## 2.2.6 Multilayer perceptron : MLP

Le MLP est un algorithme d'apprentissage supervisé qui apprend une fonction  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  en s'entraînant sur un ensemble de données, où  $n$  est le nombre de dimensions de l'entrée et  $m$  est le nombre des dimensions de sortie. Étant donné un ensemble de caractéristiques  $X := [x_1, x_2, \dots, x_m]$  et une cible  $y$ , il peut apprendre une non linéaire fonction approximatrice pour la classification ou la régression. Il est différent de la régression logistique, en ce sens qu'entre la couche d'entrée et la couche de sortie, il peut y avoir une ou plusieurs couches non linéaires, appelées couches masquées. La figure 2.7 montre une couche MLP cachée avec une sortie scalaire. [scikit learn ]

Les avantages de MLP sont :

- Capacité d'apprendre des modèles non linéaires.
- Capacité à apprendre des modèles en temps réel (apprentissage en ligne).

## 2.2.7 Benchmark

Le score physiologique aigu simplifié (SAPS) II est utilisé comme modèle de référence. SAPS II est l'un des scores d'acuité largement appliqué dans les études cliniques [Wong ].

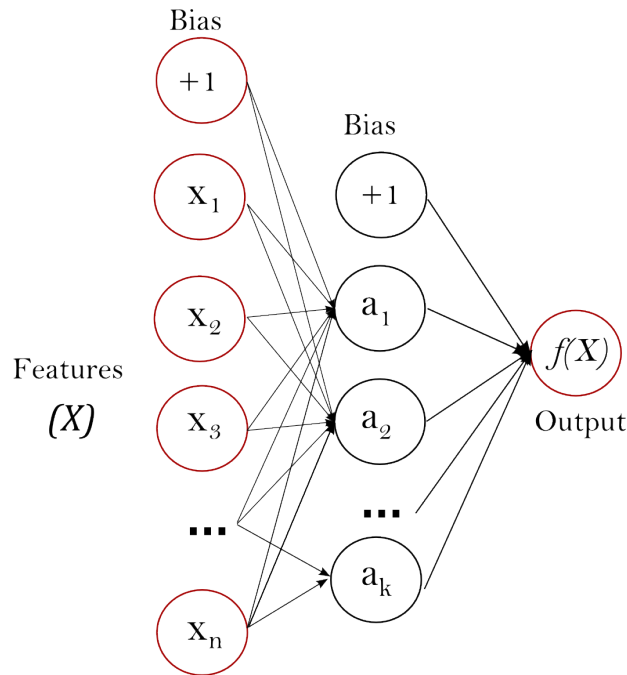


FIGURE 2.7 – Une couche MLP cachée

Il est basé sur 12 variables physiologiques dans les 24 premières heures d'admission aux soins intensifs et l'état de santé à l'admission des patients (calcul du score SAPS). Le SAPS II a été calibré pour permettre la conversion du score calculé en probabilité de mortalité. Afin de calculer les scores F1 et MCC du benchmark, un seuil est requis pour convertir la probabilité en l'étiquette prédite. Dans ce projet, nous fixons le seuil à 0,5, c'est-à-dire que si la probabilité basée sur SAPS-II est supérieure à 0,5, alors la mort à l'hôpital est prédite. En terme mathématique :  $l_{SAPS} := 1$  si  $P_{SAPS} > 0.5$ , 0 sinon.

L'étiquette de probabilité et de prédiction SAPS-II est stockée dans 'y\_test' et 'y\_train' 1.6. Les scores F1 et MCC du système de notation SAPS-II se comparent aux scores F1 et MCC des classificateurs d'apprentissage automatique.

## 2.3 Implémentation des algorithmes

1. Première tentative : DTW avec KNN : Des scripts 'dtw.py' et 'KNN.py' B personnalisés sont implémentés. 'Distance.euclidean' de la bibliothèque scipy est importé pour être utilisé comme mesure de distance pour calculer la similarité entre deux séries temporelles. De plus, les 3 voisins sont choisis pour le classificateur k-NN.
2. Seconde tentative : 'LogisticRegression', 'DecisionTreeClassifier', 'GradientBoostingClassifier' et 'KerasClassifier' sont importés de 'sklearn.linear\_model', 'sklearn.tree',

'sklearn.ensemble', et 'keras.wrappers.scikit\_learn' pour construire les apprenants correspondants.

Tout d'abord, nous avons mis en place un pipeline (est un ensemble d'éléments de traitement de données connectés en série, où la sortie d'un élément est l'entrée de l'élément suivant) pour former les quatre classificateurs aux données d'apprentissage afin d'obtenir des résultats préliminaires.

Les classificateurs par défaut sont utilisés dans 'LogisticRegression', 'DecisionTreeClassifier', 'Gradient-BoostingClassifier'. Voir Annexe B

Le code pour créer ces classificateurs est :

```
1 clf_A = LogisticRegression(random_state = 0)
2 clf_B = DecisionTreeClassifier(random_state = 0)
3 clf_C = GradientBoostingClassifier(random_state=0)
```

L'architecture MLP est montrée ici :

```
1 def keras_model():
2     clf_MLP = Sequential()
3     clf_MLP.add(Dense(128, activation="relu",
4     input_shape= data_agg["X_train"].shape[1:]))
5     clf_MLP.add(Dropout(0.1))
6     clf_MLP.add(Dense(32, activation="relu"))
7     clf_MLP.add(Dropout(0.1))
8     clf_MLP.add(Dense(8, activation="relu"))
9     clf_MLP.add(Dropout(0.1))
10    clf_MLP.add(Dense(1, activation="sigmoid"))
11    clf_MLP.compile(optimizer= "adam", loss="binary_crossentropy")
12    return clf_MLP
```

Le nombre de neurones est choisi pour rendre le nombre total de paramètres (17969 pour ce MLP) inférieur aux données d'apprentissage qui est 19220 pour réduire le risque de sur-ajustement.

# Chapitre 3

## Résultats

### 3.1 Évaluation et validation du modèle

Les données d'apprentissage contiennent 19220 échantillons et les données d'essai contiennent 4806 échantillons. Nous formons les apprenants de différentes tailles, 1%, 10% et 100% des données de formation . Nous calculons les scores en utilisant les ensembles complets de formation et de test.

### 3.2 Justification

L'arbre de décision montre le résultat le plus faible comme indiqué sur la figure [3.1](#).

L'arbre de décision donne le score F1 et le MCC le plus élevé dans les données d'apprentissage. Cependant, sa performance dans le test des données est pire que celle du benchmark (SAPS II), ce qui suggère fortement que l'apprenant de l'arbre de décision est sur-ajusté pendant le processus de formation. La régression logistique, le gradient boost et le MLP réalisent de meilleures performances que le benchmark dans le test des données, comme le montre la figure [3.1](#).

Bien que la régression logistique soit un simple classificateur, elle fonctionne mieux que le benchmark en atteignant un coefficient de corrélation de Matthews (MCC) plus élevé. De plus, en comparant les scores entre les ensembles de formation et de test, cela ne surpasse pas l'apprenant.

Le gradient boost et le MLP réalisent de bonnes améliorations et performant mieux le classificateur de régression logistique.

Le choix entre gradient boost et MLP dépend du nombre d'échantillons dans les données

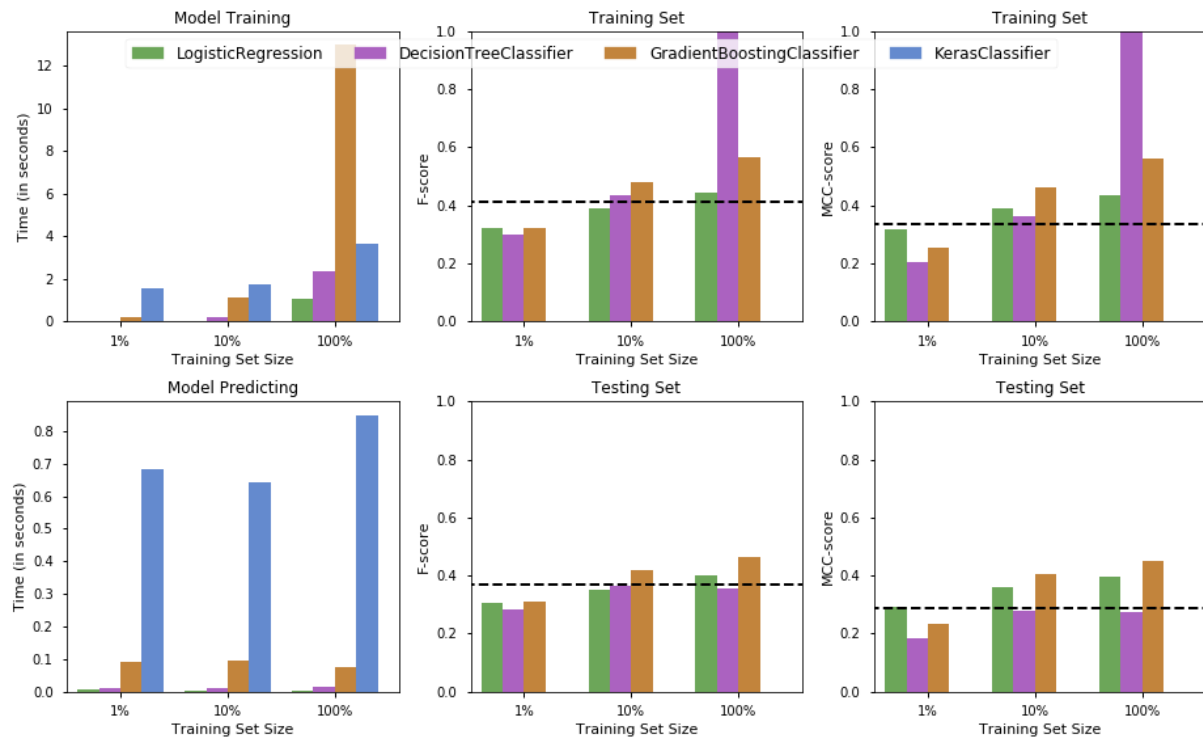


FIGURE 3.1 – Performance des métriques pour la régression logistique, l'arbre de décision, le gradient boost et le MLP. La ligne noire représente le score F1 et le MCC du SAPS II.

d'apprentissage. Lorsque les données sont suffisantes (100%), MLP peut mieux performer que le gradient boost. Cependant, il fonctionne mal lorsque la taille de l'échantillon est faible (1% ou 10%). Puisque nous pouvons supposer qu'il y a suffisamment de données médicales, le MLP pourrait avoir une meilleure position. Un des inconvénients du gradient boost et MLP est le temps d'entraînement qui est plus long. Cependant, le temps d'entraînement (<20 s dans notre jeu de données) est toujours acceptable à condition qu'il y ait une amélioration des performances de prédiction.[Wong]

Enfin, les scores F1 et MCC de prédiction de l'ensemble de données de test sont générés à l'aide du modèle MLP amélioré comme indiqué dans le Tableau 3.2 : La performance des classificateurs de gradient boost et MLP initiales et améliorés. Les scores SAPS-II dans F1 et MCC sont montrés pour la comparaison.

Classificateur	F1	MCC
SAPS II	0.371	0.290
Initial Gradient boost	0.465	0.452
Initial MLP	0.511	0.474
Tuned Gradient boost	0.484	0.448
Tuned MLP	0.485	0.447



# Chapitre 4

## Conclusion

Le but du projet est d'obtenir un meilleur pouvoir prédictif sur les décès à l'hôpital à l'unité de soins intensifs, sur la base du même ensemble de données avec le système expert médical, le score SAPS-II. Comme les classificateurs MLP et le gradient boost améliorés obtiennent des scores F1 et MCC significativement plus élevés que les scores obtenus avec le système SPAS-II, ils suggèrent que ces apprenants ont le potentiel d'être utilisés en USI pour prendre de meilleures décisions cliniques et réduire les temps d'attente pour des interventions médicales.

- Amélioration :

La variété des données (15 séries chronologiques) qui ont été extraites de la base de données MIMIC-III peut ne pas être suffisante pour atteindre une complétude élevée. Par conséquent, l'une des approches pour améliorer le pouvoir prédictif est d'utiliser le MIMIC-III complet. Après avoir obtenu une bonne prédiction, nous pouvons réduire les dimensions par diverses méthodes statistiques telles que ACP (analyse de composante principale )

- Réflexion :

MIMIC-III est une base de données riche, environ 50% du temps passé dans le projet est sur l'extraction de données et le prétraitement des données. Il y a plusieurs difficultés au cours de ce processus :

- commencer avec un minimum de connaissances en science médicale
- apprendre à écrire des scripts SQL complexes
- décider du format final des données extraites
- gérer les données manquantes

La classification des séries chronologiques multivariées n'est pas simple, et j'ai beaucoup appris pendant la résolution du problème. Au début, la revue de la littérature me suggère d'appliquer DTW avec kNN sur le problème. Cependant, au cours de la mise en œuvre, j'ai trouvé que c'était lent et le résultat n'est pas très prometteur. J'ai donc besoin de trouver une autre approche pour résoudre le problème.



# Bibliographie

- [Alistair E. W. Johnson | Roger G. Mark Alistair E. W. Johnson Tom J. Pollard.  
Reproducibility in critical care :a mortality prediction case study.
- [Elsworth | SSteven Elsworth. Dynamic Time Warping.
- [Guilloux | Agathe Guilloux. "Introduction au Machine learning M1MINT.
- [III | MIMIC III. A freely accessible critical care database. Johnson AEW, Pollard TJ, Shen L, Lehman
- [Johnson | David J. Stone Leo A. Celi Johnson Alistair EW et Tom J. Pollard.  
"The MIMIC Code Repository : enabling reproducibility in critical care research." Journal of
- [Keogh | Eamonn Keogh et Chotirat Ann Ratanamahatana.  
Everything you know about dynamic time warping is wrong.2004.
- [KNN | KNN. K plus proches voisins - [http ://webia.lip6.fr/ rifqi/COURS2001-2002/IA/knn.pdf](http://webia.lip6.fr/~rifqi/COURS2001-2002/IA/knn.pdf).
- [Mikalsen | Karl Oyvind Mikalsen. Karl Oyvind Mikalsen, Filippo Maria Bianchi, Cristina Soguero-Ru
- [Mueen | Keogh Mueen A. E. Extracting Optimal Performance from Dynamic Time Warping.
- [RL Kane | C Mueller S Duval RL Kane TA Shamliyan et T J Wilt.  
The association of registered nurse staffing levels and patient outcomes : Systematic review and
- [Sanjay Purushotham | Zhengping Che Yan Liu Sanjay Purushotham Chuizheng Meng.  
Benchmark of Deep Learning Models on Large Healthcare MIMIC Datasets.
- [scikit learn | scikit learn. [http ://scikit-learn.org/stable/modules/neural<sub>n</sub>etworks<sub>s</sub>upervised.html](http://scikit-learn.org/stable/modules/neural_networks_supervised.html) \* 0.Mallory  
Predicting Mortality for Patients in Critical Care : a Univariate Flagging Approach. Wikipedia. Ka Hu  
MLND Capstone Project : Predicting In-Hospital Mortality of ICU Patients. Jian Pei Z.Xing et Eamonn



# Annexe A

## Description des Tables utilisées

Table	Description
<b>admissions</b>	Donne des informations sur l'admission d'un patient à l'hôpital. Étant donné que chaque visite hospitalière unique pour un patient se voit attribuer un HADM_ID unique, la table peut être considérée comme une table de définition pour HADM_ID. <u>Nombre de lignes</u> : 58976 <u>Liens vers</u> : - <i>PATIENTS</i> sur <i>SUBJECT_ID</i>
<b>patients</b>	Contient toutes les données cartographiques pour tous les patients. <u>Nombre de lignes</u> : 46 520 <u>Liens vers</u> : - <i>ADMISSIONS</i> sur <i>SUBJECT_ID</i> - <i>ICUSTAYS</i> sur <i>SUBJECT_ID</i>
<b>icustays</b>	Définit chaque ICUSTAY_ID dans la base de données, c'est-à-dire définit un seul séjour USI. <u>Nombre de rangs</u> : 61 532 <u>Liens vers</u> : - <i>PATIENTS</i> sur <i>SUBJECT_ID</i> - <i>ADMISSIONS</i> sur <i>HADM_ID</i>

<b>chartevents</b>	<p>CHARTEVENTS contient toutes les données cartographiques disponibles pour un patient. Pendant leur séjour en USI, le référentiel principal de l'information d'un patient est leur carte électronique. La carte électronique affiche les signes vitaux de routine des patients et toutes les informations supplémentaires pertinentes pour leurs soins : réglages du ventilateur, valeurs du laboratoire, statut du code, état mental, etc. En conséquence, l'essentiel des informations sur le séjour d'un patient est contenu dans CHARTEVENTS. En outre, même si les valeurs de laboratoire sont capturées ailleurs (LABEVENTS), elles sont fréquemment répétées dans CHARTEVENTS. Cela se produit parce qu'il est souhaitable d'afficher les valeurs de laboratoire sur la carte électronique du patient, et ainsi les valeurs sont copiées à partir de la base de données stockant les valeurs de laboratoire dans la base de données stockant les CHARTEVENTS.</p> <p><u>Nombre de lignes</u> : 330 712 483</p> <p><u>Liens vers</u> :</p> <ul style="list-style-type: none"> <li>- <i>PATIENTS</i> sur <i>SUBJECT_ID</i></li> <li>- <i>ADMISSIONS</i> sur <i>HADM_ID</i></li> <li>- <i>ICUSTAYS</i> sur <i>ICUSTAY_ID</i></li> <li>- <i>D_ITEMS</i> sur <i>ITEMID</i></li> <li>- <i>CAREGIVERS</i> sur <i>CGID</i></li> </ul>
<b>d_items</b>	<p>Tableau de définition pour tous les éléments des bases de données de l'USI.</p> <p><u>Nombre de lignes</u> : 12 487</p> <p><u>Liens vers</u> :</p> <ul style="list-style-type: none"> <li>- <i>CHARTEVENTS</i> sur <i>ITEMID</i></li> <li>- <i>DATETIMEEVENTS</i> sur <i>ITEMID</i></li> <li>- <i>INPUTEVENTS_CV</i> sur <i>ITEMID</i></li> <li>- <i>INPUTEVENTS_MV</i> sur <i>ITEMID</i></li> <li>- <i>MICROBIOLOGYEVENTS</i> sur <i>SPEC_ITEMID</i>, <i>ORG_ITEMID</i> ou <i>AB_ITEMID</i></li> <li>- <i>OUTPUTEVENTS</i> sur <i>ITEMID</i></li> <li>- <i>PROCEDUREEVENTS_MV</i> sur <i>ITEMID</i></li> </ul>

<b>outputevents</b>	Données de sortie pour les patients. Nombre de lignes : 4 349 218 <u>Liens vers :</u> - <i>PATIENTS</i> sur <i>SUBJECT_ID</i> - <i>ADMISSIONS</i> sur <i>HADM_ID</i> - <i>ICUSTAYS</i> sur <i>ICUSTAY_ID</i> - <i>D_ITEMS</i> sur <i>ITEMID</i> - <i>CAREGIVERS</i> sur <i>CGID</i>
<b>d_labitems</b>	Tableau de définition pour toutes les mesures de laboratoire. <i>Nombre de lignes : 753</i> <u>Liens vers :</u> - <i>LABEVENTS</i> sur <i>ITEMID</i>
<b>labevents</b>	Contient toutes les mesures de laboratoire pour un patient donné, y compris les données patient. Nombre de lignes : 27 854 055 <u>Liens vers :</u> - <i>PATIENTS</i> sur <i>SUBJECT_ID</i> - <i>ADMISSIONS</i> sur <i>HADM_ID</i> - <i>D_LABITEMS</i> sur <i>ITEMID</i>
<b>SAPSII</b>	C'est un système de classification de gravité de la maladie. Son nom signifie «score de physiologie aiguë simplifiée» et est l'un des nombreux systèmes de notation ICU. Ce score est calculé à partir de 12 mesures physiologiques de routine au cours des 24 premières heures, d'informations sur l'état de santé antérieur et de certaines informations obtenues lors de l'admission.

TABLE A.1 – Description des tables utilisées





# Annexe B

## Codes : KNN & DTW

- KNN Code [Wong ] :

```
1 import numpy as np
2 from scipy.spatial import distance
3 from collections import Counter
4
5 class KNNClassifier(object):
6     """K-nearest neighbor classifier.
7     """
8     def __init__(self, n_neighbors=3, metric=distance.euclidean):
9         """Initialize the kNN object.
10
11         Args:
12             n_neighbors (int, optional): number of neighbors to use by
13             default
14             for KNN (default = 3)
15
16             metric (callable, optional): the metric used to calculate the
17             distance between two arrays (default = distance.euclidean)
18         """
19         self.n_neighbors = n_neighbors
20         self.metric = metric
21         if not callable(self.metric):
22             raise TypeError("metric must be callable")
23
24     def _cdist(self, X, Y):
25         """Computes distance between each pair of the two collections of
26         inputs
27
28         Args:
29             X (list or array): array with length of mX.
30             Y (list or array): array with length of mY.
```

```

30     Returns:
31         dm (array): A distance matrix, D. For each (i,j) element, the
32         metric(X_i, Y_j) is computed and stored in the distance matrix.
33     """
34     X = np.array(X)
35     Y = np.array(Y)
36     mX = X.shape[0]
37     mY = Y.shape[0]
38
39     # Zero matrix for starting
40     dm = np.zeros((mX, mY), dtype=np.double)
41
42     # Get the metric function
43     metric = self.metric
44
45     # Calculate the pairwise distance
46     for i in range(0, mX):
47         for j in range(0, mY):
48             dm[i, j] = metric(X[i], Y[j])
49
50     return dm
51
52 def fit(self, training_data, training_label):
53     """Fit the model by training data (training_data) and label
54     (training_label)
55
56     Args:
57         training_data (list or array): the length of list or array
58         should
59         equal the number of data points.
60         training_label (list or array): the length of list or array
61         should
62         equal the number of data points.
63     """
64     # check the dimension of training data and label
65     training_data = np.array(training_data)
66     training_label = np.array(training_label)
67     data_samples = training_data.shape[0]
68     label_samples = training_label.shape[0]
69
70     if data_samples != label_samples:
71         raise ValueError("Data and label samples must have same size.")
72     if data_samples < self.n_neighbors:
73         raise ValueError("Data size must be greater than n_neighbors.")
74     if data_samples == 0:
75         raise ValueError("Data size must be greater than 0.")
76
77     # store the data and label for future training
78     self.training_data = np.array(training_data)

```

```

77     self.training_label = np.array(training_label).reshape(-1,)
78
79     def _get_KNN_labels(self, d_matrix, n_neighbors):
80
81         # Get the indices that would sort an array.
82         sorted_indices = d_matrix.argsort()
83
84         # Get the indices for the n nearest inputs
85         KNN_indices = sorted_indices[:, :n_neighbors]
86
87         # Get the k nearest labels
88         KNN_labels = self.training_label[KNN_indices]
89
90         return KNN_labels
91
92     def predict(self, testing_data):
93         """Predict the labeling for the testing data.
94         Args:
95             testing_data (list or array): length of list or array equal the
96             number of testing data points.
97
98         Returns:
99             array: the predicted label
100         """
101         testing_data = np.array(testing_data)
102         dm = self._cdist(testing_data, self.training_data)
103         KNN_labels = self._get_KNN_labels(dm, self.n_neighbors)
104
105         voting_statistics = [Counter(KNN_label).most_common() for KNN_label
106         in KNN_labels]
107         predicted_label = [vote[0][0] for vote in voting_statistics]
108
109         return np.array(predicted_label)
110
111 if __name__ == "__main__":
112     clf = KNNClassifier(n_neighbors=3, metric=distance.euclidean)
113     print( clf.__class__.__name__)

```

- Dynamic Time Warping Code [Wong] :

```

1  """Implementation de la deformation temporelle dynamique DTW pour calculer
2  la distance de similarite entre deux series chronologiques multivariees.
3  """
4
5  import numpy as np
6  import pandas as pd
7
8  def dynamic_time_warping(a, b, metric):
9      """Returns the minimum cost between two time series, distance matrix,
10     and

```

```

9     the cost matrix.
10    Args:
11        a (list or numpy array): list/array with length n_a, which
represents
12        data points of the time series.
13        b (list or numpy array): list/array with length n_a, which
represents
14        data points of the time series.
15        metrix (callable): local distance function to calculate the distance
16        between data points in two time series.
17
18    Returns:
19        float: the minimum cost between two time series.
20        array: distance matrix
21        array: cost matrix
22    """
23    # Check the inputs
24    for x in (a,b):
25        if not (isinstance(x, list) or isinstance(x, np.ndarray)):
26            raise TypeError("input must be a list or numpy array")
27
28    # Create cost matrix with shape [n_a, n_b]
29    a, b = np.array(a), np.array(b)
30    n_a, n_b = len(a), len(b)
31    cost_matrix = np.zeros((n_a, n_b))
32
33    # Initialize the first row and column
34    cost_matrix[0,0] = metric(a[0],b[0])
35    for i in range(1, n_a):
36        cost_matrix[i, 0] = cost_matrix[i-1, 0] + metric(a[i],b[0])
37    for j in range(1, n_b):
38        cost_matrix[0, j] = cost_matrix[0, j-1] + metric(a[0], b[j])
39
40    # Calculate the subsection of cost matrix
41    for i in range(1, n_a):
42        for j in range(1, n_b):
43            cost_matrix[i, j] = min(cost_matrix[i-1, j-1], cost_matrix[i, j
-1],
44                                   cost_matrix[i-1, j]) + metric(a[i],b[j])
45
46    # Return DIW distance
47    return np.sqrt(cost_matrix[n_a-1, n_b-1]), cost_matrix[1:,1:]
48
49 def wrapping_path(cost_matrix):
50     """Return the optimal warping path. Code is based on the algorithm from
51     Chapter 4 in "Information Retrieval for Music and Motion".
52     """
53     # The last point of the optimal wrapping path
54     path = [[x-1 for x in cost_matrix.shape]]

```

---

```

55     n, m = path[-1]
56
57     while not (n == 0 and m == 0):
58         if n == 0:
59             m -= 1
60         elif m == 0:
61             n -= 1
62         else:
63             choices = [cost_matrix[n-1, m-1], cost_matrix[n-1, m],
cost_matrix[n, m-1]]
64             index = np.argmin(choices)
65             if index == 0:
66                 n, m = n-1, m-1
67             elif index == 1:
68                 n -= 1
69             else:
70                 m -= 1
71             path.append([n,m])
72     path.reverse()
73
74     return np.array(path)
75
76 def LB_Keogh(s1, s2, r):
77     LB_sum=0
78     for ind, i in enumerate(s1):
79
80         lower_bound=min(s2[(ind-r if ind-r>=0 else 0):(ind+r)])
81         upper_bound=max(s2[(ind-r if ind-r>=0 else 0):(ind+r)])
82
83         if i>upper_bound:
84             LB_sum+=(i-upper_bound)**2
85         elif i<lower_bound:
86             LB_sum+=(i-lower_bound)**2
87
88     return np.sqrt(LB_sum)
89
90 def LB_MV(a, b, r):
91     """Returns the lower bound Keogh for multivariate time series.
92     Args:
93         a (list or numpy array): list/array with length n_a, which
represents
94         data points of the time series.
95         b (list or numpy array): list/array with length n_a, which
represents
96         data points of the time series.
97         metrix (callable): local distance function to calculate the distance
98         between data points in two time series.
99
100     Returns:

```

```

101         float: the minimum cost between two time series.
102         array: distance matrix
103         array: cost matrix
104     """
105
106     LB_sum=0
107     for ind ,a_i in enumerate(a):
108         #
109         lower_bound=np.min(b[(ind-r if ind-r>=0 else 0):(ind+r)], axis=0)
110         upper_bound=np.max(b[(ind-r if ind-r>=0 else 0):(ind+r)], axis=0)
111
112         LB_sum += np.dot(a_i > upper_bound, (a_i-upper_bound)**2)
113         LB_sum += np.dot(a_i < lower_bound, (a_i-lower_bound)**2)
114
115     return np.sqrt(LB_sum)
116
117 def test():
118     import matplotlib.pyplot as plt
119     from scipy.spatial import distance
120     from timeit import default_timer as timer
121
122     metric = lambda x,y: np.sqrt(np.sum((x-y)**2))
123
124     x=np.linspace(0,4*np.pi,48)
125     ts1=np.random.rand(48,8)
126     ts2=np.random.rand(48,8)
127
128     plt.figure()
129     plt.plot(x, ts1)
130     plt.plot(x,ts2)
131
132     r = int(round(len(ts1)/10.0))
133     print(r)
134
135     start = timer()
136     print("DTW: ", dynamic_time_warping(ts1, ts2, metric)[0])
137     end = timer()
138     print('time:', end-start)
139
140
141     start = timer()
142     print("LB_MV: ", LB_MV(ts1,ts2,r))
143     end = timer()
144     print(end-start)
145
146
147 if __name__ == "__main__":
148     test()

```

## Annexe C

### Data-Extraction

# extraction

May 29, 2018

Extraction de données: Le but de ce notebook est d'extraire les données, y compris les séries temporelles multivariées de la base de données MIMIC-III. Mise en place du MIMIC-III Ce projet utilise les données de la base de données Medical Information Mart for Intensive Care (MIMIC-III). MIMIC-III contient des informations relatives aux patients admis dans les unités de soins intensifs du Beth Israel Deaconess Medical Center à Boston, Massachusetts. Avant d'extraire des données, MIMIC-III est installé dans une base de données PostgreSQL locale sous Windows. Le tutoriel sur l'installation de la base de données est disponible sur le site officiel de MIMIC-III. Puisque nous utiliserons le score SAPS-II comme référence sur la prédiction de la mortalité, nous devons calculer le score dans la base de données. Le code permettant de créer une vue matérialisée pour le score SAPS-II est également disponible dans GitHub.

```
In [5]: # Importer des bibliothèques
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import psycpg2
import os
import seaborn as sns

# pour imprimer des jolies images utilisant Pandas
from IPython.display import display, HTML

%matplotlib inline
plt.style.use('ggplot')

# les informations utilisées pour créer une connexion à la base de données, stockées en
dbinfo = "user=postgres dbname=mimic password=postgres"
```

L'extraction de données commencera par l'écriture de certaines fonctions qui seront utilisées dans ce cahier.

```
In [14]: def execute_query(query, dbinfo, params=None):
        """Exécute la requête sur la base de données locale.
        """
        Args:
        query (string): requête sql
        dbinfo (string): chaînes de connexion, par exemple,
            "user = postgres dbname = mimic password = postgres"
```



```

"""params (dictionnaire): paramètres pour les requêtes SQL
params"""

"""Résultats:
Pandas dataframe ou exception.
"""

    try:
        with psycopg2.connect(dbinfo) as con:
            # use pandas to read the query
            df = pd.read_sql_query(query, con, params=params)
            return df

    except (Exception, psycopg2.DatabaseError) as error:
        print(error)

def get_path(filename, suffix):
    """Retourne le chemin
    Args:
        filename (string): nom du fichier
        suffix (string): extension du fichier
    Résultats:
        chemin
    """
    # Créer le chemin pour le fichier
    cwd = os.getcwd()
    dir_name = os.path.join(cwd, 'data')

    return os.path.join(dir_name, filename + "." + suffix)

def save_to_csv(df, path):
    """Sauvegarde la donnée extraite dans le fichier csv sous le dossier" data ".
    Args:
        df (Pandas Dataframe): la base de données devait être sauvegardée
        filename (string): nom du fichier
    Résultats:
        Aucun
    """
    dir_name, filename = os.path.split(path)
    if not os.path.exists(dir_name):
        os.makedirs(dir_name)

    df.to_csv(path)

```

D'abord, nous pouvons plonger et obtenir une liste de toutes les tables dans MIMIC-

III. Il devrait y avoir 26 tables dans la base de données sous le nom de schéma "mimiciii" (<https://mimic.physionet.org/>).

```
In [17]: query = """select * from pg_tables where schemaname='mimiciii' and hasindexes"""

# obtenir les tables
tables = execute_query(query, dbinfo)

print("Taille de la table:", tables.shape, "\n")

print("Les 5 premières lignes de la table sont:")
tables.head()
```

Taille de la table: (43, 8)

Les 5 premières lignes de la table sont:

```
Out[17]:
```

	schemaname	tablename	tableowner	tablespace	hasindexes	hasrules	\
0	mimiciii	chartevents_1	postgres	None	True	False	
1	mimiciii	chartevents_2	postgres	None	True	False	
2	mimiciii	chartevents_3	postgres	None	True	False	
3	mimiciii	chartevents_4	postgres	None	True	False	
4	mimiciii	chartevents_5	postgres	None	True	False	

	hastriggers	rowsecurity
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False

Puisque nous nous concentrons sur les tables sous le nom de schéma "mimiciii", nous établissons un chemin de recherche pour plus de commodité.

```
In [8]: # schema name for setting a search path
schema_name = 'mimiciii'
query_schema = 'set search_path to ' + schema_name + ';'


```

Sélection de cohorte: La sélection de la cohorte est basée sur trois tableaux: patients, admissions et icustays. patients: informations sur un patient qui ne change pas - par ex. date de naissance, sexe génotypique admissions: informations enregistrées lors de l'admission à l'hôpital - type d'admission (option, urgence), heure d'admission icustays: informations enregistrées sur l'admission en unité de soins intensifs - principalement les heures d'admission et de sortie Les données que nous avons considérées lorsque ces conditions sont vraies: - Les patients ont plus de 16 ans - La durée du séjour en USI est supérieure à 1 jour mais inférieure à 10 jours - La première admission en unité de soins intensifs pendant le séjour en unité de soins intensifs - Les événements médicaux inattendus - Admissions avec des données de graphique

In [9]: *# les paramètres passeront à la requête SQL Pandas*

```
cohort_params = {'year': 60.0*60.0*24.0*365.242, 'minage': 16, 'minstay': 1, 'maxstay': 100}

cohort_query = query_schema + """
-- t0 est généré à partir de la table 'icustays'
WITH t0 AS
(
    SELECT icu.subject_id, icu.hadm_id, icu.icustay_id
    , round(cast(extract('epoch' from icu.intime - pat.dob)/%(year)s as numeric),2) AS age
    , round(cast(icu.los as numeric),2) as los
    , RANK() OVER (PARTITION BY icu.subject_id ORDER BY icu.intime) AS icustay_order
    , CASE
        WHEN round(cast(extract('epoch' from icu.intime - pat.dob)/%(year)s as numeric),2) < %(minage)s
        ELSE 0 END AS excl_age
    , CASE
        WHEN (icu.los BETWEEN %(minstay)s AND %(maxstay)s) THEN 0 ELSE 1 END AS excl_los
    , CASE
        WHEN RANK() OVER (PARTITION BY icu.subject_id ORDER BY icu.intime) != 1 THEN 1
        ELSE 0 END AS excl_icustay
    FROM icustays icu INNER JOIN patients pat ON icu.subject_id = pat.subject_id
)
-- t1 est généré à partir de la table "admissions"
, t1 AS
(
    SELECT adm.admission_type, adm.hadm_id, adm.has_chartevents_data
    , CASE
        WHEN admission_type IN ('EMERGENCY', 'URGENT') THEN 0
        ELSE 1 END AS excl_plan
    , CASE
        WHEN adm.has_chartevents_data = 1 then 0 else 1 end as excl_chartevents
    FROM admissions as adm
)
SELECT
-- les attributs
t0.subject_id, t0.hadm_id, t0.icustay_id, t0.age, t0.los,
t1.admission_type, t1.has_chartevents_data,

-- exclusions
t0.icustay_order, t0.excl_age, t0.excl_los, t0.excl_icustay,
t1.excl_plan, t1.excl_chartevents

FROM t0 INNER JOIN t1 ON t0.hadm_id = t1.hadm_id
"""
#icustay_id is the primary key in cohort table
cohort_table = execute_query(cohort_query, dbinfo, params=cohort_params)

print("Dimensions de la table de cohorte:", cohort_table.shape)
```

```
cohort_table.head()
```

Dimensions de la table de cohorte: (61532, 13)

```
Out[9]:
```

	subject_id	hadm_id	icustay_id	age	los	admission_type	\
0	58526	100001	275225	35.48	4.26	EMERGENCY	
1	54610	100003	209281	59.91	1.94	EMERGENCY	
2	9895	100006	291788	48.92	4.98	EMERGENCY	
3	23018	100007	217937	73.82	4.10	EMERGENCY	
4	533	100009	253656	60.80	2.49	EMERGENCY	

	has_chartevents_data	icustay_order	excl_age	excl_los	excl_icustay	\
0		1	1	0	0	0
1		1	1	0	0	0
2		1	1	0	0	0
3		1	1	0	0	0
4		1	1	0	0	0

	excl_plan	excl_chartevents
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

Puisque toutes les exclusions sont énumérées dans le tableau, les critères d'exclusion sont clairement énoncés. Nous résumons la sélection de la cohorte en totalisant les statistiques.

```
In [10]: # imprimer les deux premières lignes
print( '{:25s} {:5s}'.format('Condition:', 'valeur(%)') )
print( '{:25s} {:5d} ({:2.2f}%)'.format('Total des observations', cohort_table.shape[0])

# créer une liste booléenne avec df.shape [0] comme index d'exclusion
idxExcl = np.zeros(cohort_table.shape[0], dtype=bool)

# créer un dictionnaire pour stocker les données
excl_contribution = []
excl_cumulative_sum = []
columns = []

# calcule le pourcentage d'exclusion pour chaque condition
for col in cohort_table.columns:
    if "excl_" in col:
        columns.append(col)
        count = cohort_table[col].sum()
        contribution = count*100.0/cohort_table.shape[0]
        print('{:25s} {:5d} ({:2.2f}%)'.format(col, count, contribution))
        idxExcl = (idxExcl) | (cohort_table[col]==1) # au niveau du bit ou
```

```

        cumulative_sum = idxExcl.sum() # compter la somme cumulée des cas d'exclusion
        excl_contribution.append(contribution)
        excl_cumulative_sum.append(cumulative_sum/cohort_table.shape[0]*100)

# imprimer un résumé du nombre d'observations exclues en fonction des contraintes données
print('')
print('{:25s} {:5d} ({:2.2f}%)'.format('Total exclu', np.sum(idxExcl), np.sum(idxExcl)*
print('{:25s} {:5d} ({:2.2f}%)'.format('Total inclu', cohort_table.shape[0] - np.sum(id
        100 - np.sum(idxExcl)*100.0/cohort_table.shape[0]

# créer un graphique pour montrer l'exclusion
def cohort_plot(contribution, cumulative_sum, columns):
    """

    """
    """ Afficher la sélection de cohrt.
    """
    """ contributions:
    """
    """ - contirbution: (liste) la liste de la contribution indepdente de la condition d
    """
    """ - cumulative_sum: (liste) la liste de la somme cumulée des conditions données su
    """
    """ - colonnes: (liste) la liste des noms
    """
    """
    # Nombre de barres
    n = len(contribution)

    # Créer le graphique
    fig = plt.figure(figsize = (6,4))

    plt.title("Distribution sur la sélection de cohorte", fontsize = 16)
    plt.bar(np.arange(n), contribution, width = 0.6, align="center", color = '#648ace',
            label = "Indépendant")
    plt.bar(np.arange(n) - 0.3, cumulative_sum, width = 0.2, align = "center", color =
            label = "Cumulatif")
    plt.xticks(np.arange(n), columns)
    plt.xlim((-0.5, n-0.5))
    plt.ylabel("Exclusion (%)", fontsize = 12)
    plt.xlabel("Condition d'exclusion", fontsize = 12)
    plt.legend(loc = 'upper left')
    if not os.path.exists('plot'):
        os.makedirs('plot')
    plt.savefig('plot/distribution_sélection_cohorte')

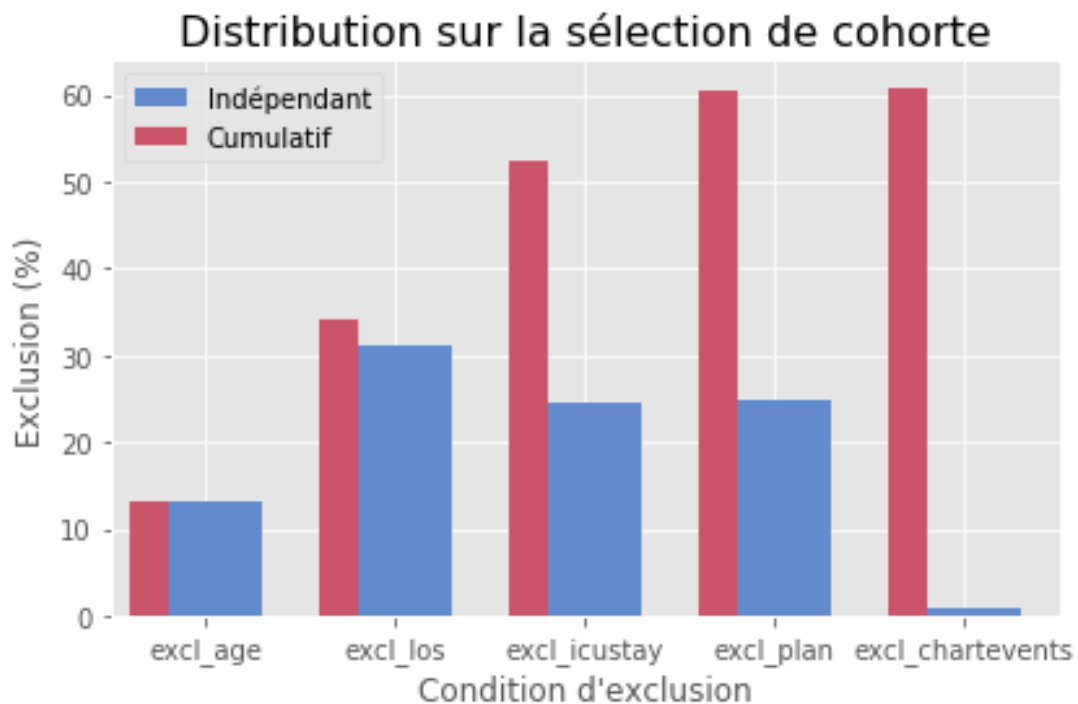
    plt.tight_layout()
    plt.show()

cohort_plot(excl_contribution, excl_cumulative_sum, columns)

```

Condition:                      valeur(%)

Total des observations	61532 (100.00%)
excl_age	8109 (13.18%)
excl_los	19089 (31.02%)
excl_icustay	15056 (24.47%)
excl_plan	15331 (24.92%)
excl_chartevents	481 (0.78%)
 Total exclu	 37495 (60.94%)
Total inclu	24037 (39.06%)



Au total, environ 61% des patients sont exclus pour les raisons que nous avons mentionnées. Nous pouvons maintenant obtenir les identifiants de séjour de la cohorte, qui seront utilisés pour l'exploration et l'analyse des données.

```
In [11]: # rejoindre toutes les conditions d'exclusion
conditions = (cohort_table['excl_age']==0) & (cohort_table['excl_los']==0) \
            & (cohort_table['excl_icustay']==0) & (cohort_table['excl_plan']==0) \
            & (cohort_table['excl_chartevents']==0)

# Sélectionnez la cohorte en fonction des conditions de jointure donnant
cohort = cohort_table[conditions]

# obtenir les cohortID
cohortIDs = tuple([x for x in cohort['icustay_id']]) # cast to integer
```

```
print( 'Total des observations dans la cohorte:', cohort.shape[0])
cohort.head()
```

Total des observations dans la cohorte: 24037

```
Out[11]:
```

	subject_id	hadm_id	icustay_id	age	los	admission_type	\
0	58526	100001	275225	35.48	4.26	EMERGENCY	
1	54610	100003	209281	59.91	1.94	EMERGENCY	
2	9895	100006	291788	48.92	4.98	EMERGENCY	
3	23018	100007	217937	73.82	4.10	EMERGENCY	
4	533	100009	253656	60.80	2.49	EMERGENCY	

	has_chartevents_data	icustay_order	excl_age	excl_los	excl_icustay	\
0	1	1	0	0	0	
1	1	1	0	0	0	
2	1	1	0	0	0	
3	1	1	0	0	0	
4	1	1	0	0	0	

	excl_plan	excl_chartevents
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

Démographie de la cohorte Les détails généraux des patients peuvent être extraits des tableaux: patients et icustays. Le sexe, l'âge, le temps, le temps passé, la durée du séjour et la survie sont considérés. De plus, le décès à l'hôpital est également calculé.

```
In [20]: # définir les paramètres
params = {'year': 60.0*60.0*24.0*365.242, 'day': 60.0*60.0*24.0, 'cohortIDs':cohortIDs}

# écrire la requête
query = query_schema + """
SELECT icu.subject_id, icu.hadm_id, icu.icustay_id, pat.gender, icu.dbsource
, ROUND(CAST(EXTRACT(epoch FROM icu.intime - pat.dob)/%(year)s AS numeric),2) AS age
, icu.intime
, round(cast(extract(epoch from adm.disctime - icu.intime)/%(day)s as numeric), 2) as
, case
    when (pat.dod is not null)
    then round(cast(extract(epoch from pat.dod - icu.intime)/%(day)s as numeric), 2)
    else -1 end as survival
, case
    when round(cast(extract(epoch from adm.disctime - icu.intime)/%(day)s as numeric),
        >= round(cast(extract(epoch from pat.dod - icu.intime)/%(day)s as numeric), 2)
    then 1 else 0 end as in_hospital_death
FROM icustays icu
```

```

INNER JOIN patients pat ON icu.subject_id = pat.subject_id
INNER JOIN admissions adm on adm.hadm_id = icu.hadm_id
WHERE icu.icustay_id in %(cohortIDs)s
ORDER BY icu.subject_id, icu.intime;
"""
# exécuter la requête
patient_details = execute_query(query, dbinfo, params=params)
print("Dimensions de la table des détails des patients:", patient_details.shape)

patient_details.head()

```

Dimensions de la table des détails des patients: (24037, 10)

```

Out[20]:
  subject_id  hadm_id  icustay_id  gender  dbsource   age      intime \
0          3   145834    211552      M   carevue  76.53  2101-10-20 19:10:11
1          4   185777    294638      F   carevue  47.85  2191-03-16 00:29:31
2          9   150750    220597      M   carevue  41.79  2149-11-09 13:07:02
3         11   194540    229441      F   carevue  50.15  2178-04-16 06:19:32
4         13   143045    263738      F   carevue  39.87  2167-01-08 18:44:25

   los  survival  in_hospital_death
0  10.78    236.20                 0
1   7.76    -1.00                 0
2   4.88     4.45                 1
3  25.53   211.74                 0
4   6.85    -1.00                 0

```

Pour protéger la confidentialité du patient, toutes les dates dans la base de données ont été décalées. De plus, la date de naissance du patient a été décalée avant l'année 1900 si le patient est âgé de plus de 89 ans. Dans ces cas, l'âge du patient à la première admission a été fixé à 300 ans. L'âge médian pour ce groupe de patients est de 91,4 (<https://mimic.physionet.org/mimictables/patients/>). Par conséquent, l'âge des patients de plus de 89 ans est remplacé par la médiane, qui est de 91,4 ans.

```

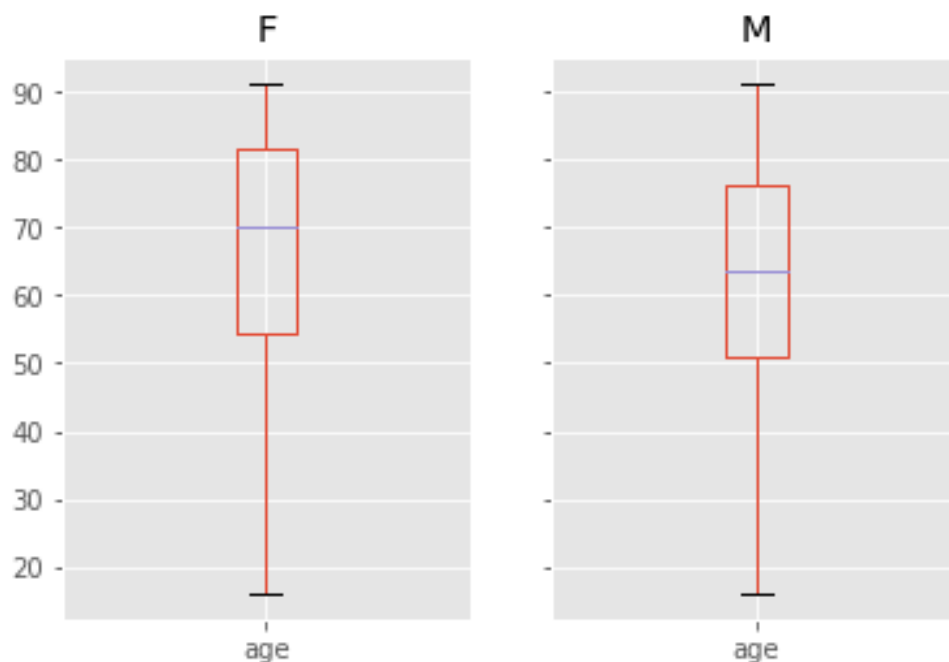
In [21]: # identifier le groupe dans lequel leurs âges sont décalés
shifted_age = patient_details['age'] >= 300

# remplacer l'âge avec 91,4
patient_details.loc[shifted_age, 'age'] = 91.4

# diagramme en boîte
age_bplot = patient_details[['gender', 'age']].groupby('gender').boxplot(return_type='ax')

```





Dans la section suivante, la taille et le poids moyens des patients sont extraits des tableaux: charttime et icustay. La requête SQL est modifiée à partir de la requête SQL à partir du code dans GitHub.

In [22]: *# the query to get the height and weight*

```
query = query_schema + """
with rawdata as
( -- extraction, transformation et nettoyage de données
SELECT c.charttime, c.itemid,c.subject_id, c.icustay_id,
CASE
    WHEN c.itemid IN (762, 763, 3723, 3580, 3581, 3582, 226512) THEN 'WEIGHT'
    WHEN c.itemid IN (920, 1394, 4187, 3486, 3485, 4188, 226707) THEN 'HEIGHT'
END AS category,
-- Assurer que tous les poids sont en kg et que les hauteurs sont en centimètres
CASE
    WHEN c.itemid IN (3581, 226531) THEN c.valuenum * 0.45359237          -- lb e
    WHEN c.itemid IN (3582) THEN c.valuenum * 0.0283495231             -- once
    WHEN c.itemid IN (920, 1394, 4187, 3486, 226707) THEN c.valuenum * 2.54 -- in e
    ELSE c.valuenum
END AS valuenum
FROM chartevents c
WHERE c.icustay_id in %(cohortIDs)s
    AND c.valuenum IS NOT NULL
    -- exclude rows marked as error
    AND c.error IS DISTINCT FROM 1 -- not 1
    AND c.itemid IN (-- Carevue
```

```

762, 763, 3723, 3580, -- Poids Kg
3581, -- Poids lb
3582, -- Poids oz
920, 1394, 4187, 3486, -- taille inch
3485, 4188, -- taille cm
-- Metavision
226707, -- taille, cm
226512) -- Poids d'ad

AND c.valuenum <> 0
)
, singlefeature as
( -- feature selection
SELECT distinct subject_id, icustay_id, category,
avg(valuenum) over
(partition BY subject_id, icustay_id, category
order by charttime ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS
FROM rawdata order by subject_id, icustay_id, category
)
, pivotfeatures AS
(
SELECT subject_id, icustay_id,
MAX(case when category = 'HEIGHT' then avg_valuenum else NULL end) AS height_avg,
MAX(case when category = 'WEIGHT' then avg_valuenum else NULL end) AS weight_avg
FROM singlefeature GROUP BY subject_id, icustay_id
)

SELECT f.icustay_id, f.subject_id,
ROUND(cast(f.height_avg as numeric),1) AS height_avg,
ROUND(cast(f.weight_avg as numeric),1) AS weight_avg
FROM pivotfeatures f ORDER BY subject_id, icustay_id;
"""

patient_weightheight = execute_query(query, dbinfo, params=params)
patient_weightheight.head()

```

```

Out[22]:   icustay_id  subject_id  height_avg  weight_avg
0         211552           3         179.1         101.9
1         294638           4           NaN          53.6
2         220597           9         182.9         102.9
3         263738          13         144.8          73.6
4         217847          21         175.3          65.3

```

Ensuite, la relation poids / taille du patient est concaténée dans la relation détail patient. La relation résultante est enregistrée pour une utilisation future.

```

In [11]: # Pour concaténer les tables patient_details et patient_weightheight
cols_to_use = patient_weightheight.columns.difference(patient_details.columns)
patient_details = pd.concat([patient_details, patient_weightheight[cols_to_use]], axis=

```

```

# Pour enregistrer l'élément de données dans 'patient_details'
file_path = get_path('patient_details', 'csv')
if not os.path.exists('data'):
    os.makedirs('data')
save_to_csv(patient_details, file_path)
print("Dimensions de la table concaténée :", patient_details.shape)

# Afficher la table concaténée
patient_details.head()

```

Dimensions de la table concaténée : (24037, 12)

```

Out[11]:
  subject_id  hadm_id  icustay_id  gender  dbsource   age      intime \
0           3   145834   211552      M  carevue  76.53  2101-10-20  19:10:11
1           4   185777   294638      F  carevue  47.85  2191-03-16   00:29:31
2           9   150750   220597      M  carevue  41.79  2149-11-09   13:07:02
3          11   194540   229441      F  carevue  50.15  2178-04-16   06:19:32
4          13   143045   263738      F  carevue  39.87  2167-01-08   18:44:25

      los  survival  in_hospital_death  height_avg  weight_avg
0   10.78    236.20                   0      179.1      101.9
1    7.76     -1.00                   0         NaN      53.6
2    4.88     4.45                   1      182.9      102.9
3   25.53    211.74                   0      144.8      73.6
4    6.85     -1.00                   0      175.3      65.3

```

Mesures physiologiques: Les mesures physiologiques liées au score de SAPS-II ont été choisies pour caractériser l'état du patient. Ces mesures sont: fréquence cardiaque, pression artérielle systolique, fréquence respiratoire, température, GCS, urine, nombre de globules blancs, azote uréique sanguin, potassium, sodium, bicarbonate, pression partielle d'oxygène et fraction d'oxygène inspiré. La fréquence cardiaque, la fréquence respiratoire, la pression artérielle systolique, la GCS et la température sont enregistrées dans le tableau des caractéristiques. La sortie d'urine est enregistrée dans la table outputevents. Les globules blancs, l'azote uréique du sang, le potassium, le sodium, le bicarbonate, la pression partielle d'oxygène et la fraction d'oxygène inspiré sont enregistrés dans le tableau des laves. Nous commençons l'extraction des données de la table chartevents. La mesure est échantillonnée de manière irrégulière lorsque le patient est en réanimation, mais nous considérons uniquement les données générées à partir du premier jour de séjour en unité de soins intensifs car il est plus important d'identifier le patient à haut risque dans son séjour précoce en réanimation.

```

In [12]: # définir les paramètres : les éléments en chartevents, outputevents, et labevents sont
# les données du graphique ont deux sources, carevue et métavision. Les deux sont consi
params = {'chartitems':(211,      # Heart rate
                    220045,      # Heart rate
                    51,         # Arterial BP [Systolic]
                    442,        # Manual BP [Systolic]
                    455,        # NBP [Systolic]

```

```

        6701,      # Arterial BP [Systolic]
220179,      # Non Invasive Blood Pressure systolic
220050,      # Arterial Blood Pressure systolic
        676,      # Temperature C
        677,      # Temperature C
        678,      # Temperature F
        679,      # Temperature F calc
223762,      # Temperature C
223761,      # Temperature F
        723,      # GCSVerbal
        454,      # GCSEyes
        184,      # GCSMotor
223900,      # GCS - Verbal Response
223901,      # GCS - Motor Response
220739,      # GCS - Eye Opening
        618,      # Respiratory Rate
        615,      # Resp Rate (Total)
220210,      # Respiratory Rate
224690      # Respiratory Rate (Total)
    ),
    'outputitems':(
        40055, # "Urine Out Foley"
        43175, # "Urine ."
        40069, # "Urine Out Void"
        40094, # "Urine Out Condom Cath"
        40715, # "Urine Out Suprapubic"
        40473, # "Urine Out IleoConduit"
        40085, # "Urine Out Incontinent"
        40057, # "Urine Out Rt Nephrostomy"
        40056, # "Urine Out Lt Nephrostomy"
        40405, # "Urine Out Other"
        40428, # "Urine Out Straight Cath"
        40086, # Urine Out Incontinent
        40096, # "Urine Out Ureteral Stent #1"
        40651, # "Urine Out Ureteral Stent #2"
        226559, # "Foley"
        226560, # "Void"
        226561, # "Condom Cath"
        226584, # "Ileoconduit"
        226563, # "Suprapubic"
        226564, # "R Nephrostomy"
        226565, # "L Nephrostomy"
        226567, # Straight Cath
        226557, # R Ureteral Stent
        226558, # L Ureteral Stent
        227489 # GU Irrigant/Urine Volume Out
    ),
    'labitems':(

```

```

51300, # WBC
51301, # WBC
51006, # BUN
50822, # Potassium
50971, # Potassium
50983, # Sodium
50824, # Sodium
50882, # HCO3
50821, # PO2
50816, # FIO2
50885 # 'BILIRUBIN'
),
'cohortIDs': cohortIDs}

```

Extraction de données de la table chartevents

```

In [20]: query = query_schema + """
with cohort as
( -- select the ICU stay IDs of the cohort
select icu.icustay_id, icu.intime FROM icustays icu
where icu.icustay_id in %(cohortIDs)s
), rawdata as
(-- only the data in the first day of ICU stay is considered
select c.icustay_id, cohort.intime, c.charttime,
cast(c.charttime - cohort.intime as interval) as time,
c.itemid, c.valuenum
from chartevents c inner join cohort on cohort.icustay_id = c.icustay_id
where c.charttime between cohort.intime and cohort.intime + INTERVAL '1 day'
and c.itemid in %(chartitems)s
and c.valuenum IS NOT NULL
and c.error IS DISTINCT FROM 1
--and c.valuenum <> 0.0
), combination as
(-- combine data
select rd.icustay_id, rd.intime, rd.time, rd.itemid,
case
when rd.itemid in (211, 220045) then 'heart rate'
when rd.itemid in (51, 442, 455, 6701, 220179, 220050) then 'sys_BP'
when rd.itemid in (723, 223900, 454, 223901, 184, 220739) then 'GCS'
when rd.itemid in (223762, 676, 677, 223761, 678, 679) then 'temperature'
when rd.itemid in (618, 615, 220210, 224690) then 'resp_rate'
--when rd.itemid in (646, 220277) then 'spO2'
end as category,
case
when rd.itemid in (223762, 676, 677) and rd.valuenum > 20 and rd.valuenum < 45 then
when rd.itemid in (223761, 678, 679) and rd.valuenum > 68 and rd.valuenum < 113 the
when rd.itemid in (211, 220045) and rd.valuenum > 0 and rd.valuenum < 250 then rd.v
when rd.itemid in (51, 442, 455, 6701, 220179, 220050) and rd.valuenum > 0 and rd.v

```

```

        when rd.itemid in (723, 223900, 454, 223901, 184, 220739) and rd.valuenum >= 1 and
        when rd.itemid in (618, 615, 220210, 224690) and rd.valuenum > 0 and rd.valuenum <
        else null
    end as valuenum
from rawdata rd
where rd.itemid in %(chartitems)s
)
select comb.icustay_id, comb.intime, comb.time, comb.category,
round(cast(avg(comb.valuenum) as numeric),2) as valuenum
from combination comb group by comb.icustay_id, comb.intime, comb.time, comb.category
order by comb.icustay_id, comb.intime, comb.time, comb.category
"""

```

*# Exécuter la requête*

```
chartdata = execute_query(query, dbinfo, params=params)
```

*# Enregistrer les données au format CSV*

```
file_path = get_path('chartdata', 'csv')
```

```
if not os.path.exists('data'):
```

```
    os.makedirs('data')
```

```
save_to_csv(chartdata, file_path)
```

*# Afficher les premières lignes du dataframe*

```
chartdata.head()
```

```
Out[20]:
```

	icustay_id	intime	time	category	valuenum
0	200003	2199-08-02 19:50:04	00:09:56	GCS	5.0
1	200003	2199-08-02 19:50:04	00:09:56	heart rate	119.0
2	200003	2199-08-02 19:50:04	00:09:56	resp_rate	35.0
3	200003	2199-08-02 19:50:04	00:09:56	sys_BP	91.0
4	200003	2199-08-02 19:50:04	01:09:56	heart rate	122.0

print(artdata) Extraction de données à partir de la table labevents

```
In [21]: query = query_schema + """
```

```
with cohort as
```

```
(-- sélectionnez les ID de séjour de l'unité de soins intensifs de la cohorte
```

```
select icu.icustay_id, icu.hadm_id, icu.intime FROM icustays icu
```

```
where icu.icustay_id in %(cohortIDs)s
```

```
), rawdata as
```

```
(
```

```
select cohort.icustay_id, cohort.intime, lab.charttime,
```

```
cast(lab.charttime - cohort.intime as interval) as time,
```

```
lab.itemid, lab.valuenum
```

```
from labevents lab inner join cohort on cohort.hadm_id = lab.hadm_id
```

```
where lab.charttime between (cohort.intime - interval '6' hour) and cohort.intime + INT
```

```
and lab.itemid in %(labitems)s
```

```

        and lab.valuenum IS NOT NULL
        --and lab.valuenum <> 0.0
    ), combination as
    (
select rd.icustay_id, rd.intime, rd.time, rd.itemid,
case
    -- when rd.itemid in (50820) then 'pH'
    when rd.itemid in (51300, 51301) then 'WBC'
    when rd.itemid in (51006) then 'BUN'
    when rd.itemid in (50822, 50971) then 'potassium'
    when rd.itemid in (50983, 50824) then 'sodium'
    when rd.itemid in (50882) then 'HCO3'
    when rd.itemid in (50821) then 'PO2'
    when rd.itemid in (50816) then 'FIO2'
    when rd.itemid in (50885) then 'BILIRUBIN'
end as category,
case
    when rd.itemid in (51300, 51301) and rd.valuenum > 0 and rd.valuenum < 1000 then rd.valuenum
    when rd.itemid in (51006) and rd.valuenum > 0 and rd.valuenum < 300 then rd.valuenum
    when rd.itemid in (50822, 50971) and rd.valuenum > 0 and rd.valuenum < 30 then rd.valuenum
    when rd.itemid in (50983, 50824) and rd.valuenum > 0 and rd.valuenum < 200 then rd.valuenum
    when rd.itemid in (50882) and rd.valuenum > 0 and rd.valuenum < 10000 then rd.valuenum
    when rd.itemid in (50821) and rd.valuenum > 0 and rd.valuenum < 800 then rd.valuenum
    when rd.itemid in (50816) and rd.valuenum > 0 and rd.valuenum < 100 then rd.valuenum
    when rd.itemid in (50885) and rd.valuenum < 150 then rd.valuenum
else null end as valuenum
from rawdata rd
where rd.itemid in %(labitems)s
    )
select comb.icustay_id, comb.intime, comb.time, comb.category,
round(cast(avg(comb.valuenum) as numeric),2) as valuenum
from combination comb group by comb.icustay_id, comb.intime, comb.time, comb.category
order by comb.icustay_id, comb.intime, comb.time, comb.category
"""

# Exécuter la requête
labdata = execute_query(query, dbinfo, params=params)

# Enregistrer les données au format CSV
file_path = get_path('labdata', 'csv')
if not os.path.exists('data'):
    os.makedirs('data')
save_to_csv(labdata, file_path)

# Imprimer les premières lignes de données
labdata.head()

```

```

Out[21]:      icustay_id      intime      time  category  valuenum
0      200003  2199-08-02  19:50:04 -1 days +21:49:56  BILIRUBIN      3.5

```

1	200003	2199-08-02	19:50:04	-1 days +21:49:56	BUN	21.0
2	200003	2199-08-02	19:50:04	-1 days +21:49:56	HCO3	23.0
3	200003	2199-08-02	19:50:04	-1 days +21:49:56	WBC	14.8
4	200003	2199-08-02	19:50:04	-1 days +21:49:56	potassium	3.1

In [19]: `print(labdata)`

	icustay_id	intime	time	category	valuenum
0	200003	2199-08-02 19:50:04	-1 days +21:49:56	BILIRUBIN	3.5
1	200003	2199-08-02 19:50:04	-1 days +21:49:56	BUN	21.0
2	200003	2199-08-02 19:50:04	-1 days +21:49:56	HCO3	23.0
3	200003	2199-08-02 19:50:04	-1 days +21:49:56	WBC	14.8
4	200003	2199-08-02 19:50:04	-1 days +21:49:56	potassium	3.1
5	200003	2199-08-02 19:50:04	-1 days +21:49:56	sodium	140.0
6	200003	2199-08-02 19:50:04	05:56:56	BILIRUBIN	3.4
7	200003	2199-08-02 19:50:04	05:56:56	BUN	20.0
8	200003	2199-08-02 19:50:04	05:56:56	HCO3	18.0
9	200003	2199-08-02 19:50:04	05:56:56	WBC	40.2
10	200003	2199-08-02 19:50:04	05:56:56	potassium	3.2
11	200003	2199-08-02 19:50:04	05:56:56	sodium	141.0
12	200003	2199-08-02 19:50:04	06:04:56	PO2	88.0
13	200003	2199-08-02 19:50:04	07:51:56	PO2	94.0
14	200003	2199-08-02 19:50:04	14:58:56	PO2	108.0
15	200003	2199-08-02 19:50:04	21:46:56	BILIRUBIN	3.9
16	200003	2199-08-02 19:50:04	21:46:56	BUN	19.0
17	200003	2199-08-02 19:50:04	21:46:56	HCO3	25.0
18	200003	2199-08-02 19:50:04	21:46:56	WBC	43.9
19	200003	2199-08-02 19:50:04	21:46:56	potassium	3.1
20	200003	2199-08-02 19:50:04	21:46:56	sodium	144.0
21	200003	2199-08-02 19:50:04	22:01:56	PO2	74.0
22	200003	2199-08-02 19:50:04	22:49:56	PO2	163.0
23	200007	2109-02-17 10:03:37	-1 days +22:40:23	BUN	13.0
24	200007	2109-02-17 10:03:37	-1 days +22:40:23	HCO3	23.0
25	200007	2109-02-17 10:03:37	-1 days +22:40:23	WBC	7.9
26	200007	2109-02-17 10:03:37	-1 days +22:40:23	potassium	4.5
27	200007	2109-02-17 10:03:37	-1 days +22:40:23	sodium	140.0
28	200007	2109-02-17 10:03:37	10:56:23	BUN	10.0
29	200007	2109-02-17 10:03:37	10:56:23	HCO3	22.0
...	...	...	...	...	...
457405	299993	2149-11-13 21:01:05	03:35:55	PO2	70.0
457406	299993	2149-11-13 21:01:05	08:58:55	BILIRUBIN	0.5
457407	299993	2149-11-13 21:01:05	08:58:55	BUN	13.0
457408	299993	2149-11-13 21:01:05	08:58:55	HCO3	31.0
457409	299993	2149-11-13 21:01:05	08:58:55	WBC	12.2
457410	299993	2149-11-13 21:01:05	08:58:55	potassium	4.2
457411	299993	2149-11-13 21:01:05	08:58:55	sodium	135.0
457412	299993	2149-11-13 21:01:05	18:53:55	BUN	12.0
457413	299993	2149-11-13 21:01:05	18:53:55	HCO3	30.0



457414	299993	2149-11-13	21:01:05	18:53:55	potassium	3.4
457415	299993	2149-11-13	21:01:05	18:53:55	sodium	136.0
457416	299995	2116-03-04	17:44:39	-1 days +20:09:21	BILIRUBIN	0.3
457417	299995	2116-03-04	17:44:39	-1 days +20:09:21	BUN	10.0
457418	299995	2116-03-04	17:44:39	-1 days +20:09:21	HCO3	25.0
457419	299995	2116-03-04	17:44:39	-1 days +20:09:21	WBC	20.8
457420	299995	2116-03-04	17:44:39	-1 days +20:09:21	potassium	3.8
457421	299995	2116-03-04	17:44:39	-1 days +20:09:21	sodium	140.0
457422	299995	2116-03-04	17:44:39	-1 days +20:19:21	potassium	3.6
457423	299995	2116-03-04	17:44:39	-1 days +20:19:21	sodium	139.0
457424	299995	2116-03-04	17:44:39	-1 days +20:26:21	FIO2	NaN
457425	299995	2116-03-04	17:44:39	-1 days +20:26:21	PO2	145.0
457426	299995	2116-03-04	17:44:39	06:17:21	PO2	170.0
457427	299995	2116-03-04	17:44:39	06:17:21	potassium	3.3
457428	299995	2116-03-04	17:44:39	08:48:21	BUN	6.0
457429	299995	2116-03-04	17:44:39	08:48:21	HCO3	25.0
457430	299995	2116-03-04	17:44:39	08:48:21	WBC	24.2
457431	299995	2116-03-04	17:44:39	08:48:21	potassium	3.8
457432	299995	2116-03-04	17:44:39	08:48:21	sodium	140.0
457433	299995	2116-03-04	17:44:39	09:16:21	PO2	116.0
457434	299995	2116-03-04	17:44:39	09:16:21	potassium	3.8

[457435 rows x 5 columns]

## Extraire le SAPS-II

```
In [17]: query = query_schema + """
select icustay_id, sapsii_prob
from SAPSII where icustay_id in %(cohortIDs)s
"""

# Exécuter la requête
sapsii = execute_query(query, dbinfo, params=params)

# Enregistrer les données SAPS-II
file_path = get_path('sapsii_score', 'csv')
if not os.path.exists('data'):
    os.makedirs('data')
save_to_csv(sapsii, file_path)

# Afficher les premières lignes des données
sapsii.head()
```

```
Out[17]:   icustay_id  sapsii_prob
0      200003      0.106398
1      200007      0.029295
2      200014      0.305597
```

3	200019	0.552904
4	200021	0.326364

## Annexe D

### Data-Wrangling

# Data-wrangling

May 29, 2018

Data wrangling Dans ce cahier, nous prétraiterons les jeux de données extraits de la base de données MIMIC-III.

```
In [31]: # Importer les bibliothèques
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import datetime
import seaborn as sns
import urllib.request, urllib.parse, urllib.error
import zipfile

# pour avoir des jolies pandas dataframes
from IPython.display import display, HTML

%matplotlib inline
plt.style.use('seaborn-notebook')

In [12]: # générer un dossier de données, nous stockons toutes les données dans ce dossier
if not os.path.exists('data'):
    os.makedirs('data')

# téléchargez et décompressez les rawdata
url = 'https://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly8xZHJ2Lm1zL3UvcyFBdk96S3k0djAyd
urllib.request.urlretrieve(url, "download.zip")

with zipfile.ZipFile("download.zip", "r") as zip_ref:
    zip_ref.extractall("data/rawdata")
```

Collecte de données Les jeux de données sont stockés au format CSV dans le dossier 'data / rawdata'.

```
In [13]: def get_path(filename, suffix):
        """Return the path

        Args:
            filename (string): name of the file
```

*suffix (string): extension of the file*

*Returns:*

*path*

*"""*

*# Joindre le chemin du dossier "data"*

`cwd = os.getcwd()`

`dir_name = os.path.join(cwd, 'data/rawdata')`

`return os.path.join(dir_name, filename + "." + suffix)`

Recueillir les données de caractéristiques qui sont les séries temporelles multivariées Les fonctionnalités sont stockées dans les fichiers chartdata, labdata et outputdata CSV. D'abord, nous chargeons les données de ces trois fichiers.

In [14]: *# obtenir les noms de fichiers*

`filenames = ['chartdata', 'labdata', 'outputdata']`

`paths = [get_path(filename, 'csv') for filename in filenames]`

*# Lire et stocker des données dans une liste*

`rawdata = []`

`for path in paths:`

`df = pd.read_csv(path, index_col=0, converters={'intime': np.datetime64}, engine='py`

`df.loc[:, 'time'] = pd.to_timedelta(df.loc[:, 'time']) # cast the dtype to timedelta`

`rawdata.append(df)`

Imprimez les informations des données et les premières lignes des données.

In [15]: *# imprimer les types de données*

`for i, filename in enumerate(filenames):`

`print("Les types de données dans {} sont: \n".format(filename), rawdata[i].dtypes,`

`print("Les 5 premières lignes de {} sont:\n".format(filename), rawdata[i].head(), "`

Les types de données dans chartdata sont:

icustay\_id int64

intime datetime64[ns]

time timedelta64[ns]

category object

valuenum float64

dtype: object

Les 5 premières lignes de chartdata sont:

	icustay_id	intime	time	category	valuenum
0	200003	2199-08-02 19:50:04	00:09:56	GCS	5.0
1	200003	2199-08-02 19:50:04	00:09:56	heart rate	119.0
2	200003	2199-08-02 19:50:04	00:09:56	resp_rate	35.0
3	200003	2199-08-02 19:50:04	00:09:56	sys_BP	91.0
4	200003	2199-08-02 19:50:04	01:09:56	heart rate	122.0

Les types de données dans labdata sont:

```
icustay_id      int64
intime          datetime64[ns]
time            timedelta64[ns]
category        object
valuenum        float64
dtype: object
```

Les 5 premières lignes de labdata sont:

	icustay_id	intime	time	category	valuenum
0	200003	2199-08-02 19:50:04 -1 days +21:49:56	BILIRUBIN	3.5	
1	200003	2199-08-02 19:50:04 -1 days +21:49:56	BUN	21.0	
2	200003	2199-08-02 19:50:04 -1 days +21:49:56	HCO3	23.0	
3	200003	2199-08-02 19:50:04 -1 days +21:49:56	potassium	3.1	
4	200003	2199-08-02 19:50:04 -1 days +21:49:56	sodium	140.0	

Les types de données dans outputdata sont:

```
icustay_id      int64
intime          datetime64[ns]
time            timedelta64[ns]
category        object
valuenum        float64
dtype: object
```

Les 5 premières lignes de outputdata sont:

	icustay_id	intime	time	category	valuenum
0	200003	2199-08-02 19:50:04 00:24:56	urine_out	230.0	
1	200003	2199-08-02 19:50:04 02:09:56	urine_out	NaN	
2	200003	2199-08-02 19:50:04 03:09:56	urine_out	90.0	
3	200003	2199-08-02 19:50:04 04:09:56	urine_out	37.0	
4	200003	2199-08-02 19:50:04 05:09:56	urine_out	220.0	

Nous concaténons ces dataframes selon l'axe = 0. En outre, nous convertissons la colonne 'time' du type timedelta en ajoutant la date actuelle.

In [16]: *# concaténer les rawdata*

```
df = pd.concat(rawdata)
```

```
df['time'] = df['time'] + datetime.date.today() # convert the timedelta to datetime
```

```
# Imprimer le fichier de données résultant
```

```
df.head()
```

```
Out[16]:
```

	icustay_id	intime	time	category	valuenum
0	200003	2199-08-02 19:50:04 2018-05-25 00:09:56	GCS	5.0	
1	200003	2199-08-02 19:50:04 2018-05-25 00:09:56	heart rate	119.0	
2	200003	2199-08-02 19:50:04 2018-05-25 00:09:56	resp_rate	35.0	

3	200003	2199-08-02	19:50:04	2018-05-25	00:09:56	sys_BP	91.0
4	200003	2199-08-02	19:50:04	2018-05-25	01:09:56	heart rate	122.0

Nous faisons ensuite pivoter la table en faisant passer les colonnes en catégories de telle sorte que chaque rangée montre toutes les variables d'une observation, ce qui est un peu plus près des données ordonnées.

```
In [17]: # appliquer le pivot à la df
feature_df = df.pivot_table('valuenum', index=['icustay_id', 'time'], columns='category')
feature_df = feature_df.sort_index()
feature_df.head()
```

```
Out[17]: category          BILIRUBIN  BUN  FIO2  GCS  HC03  P02  WBC  \
icustay_id time
200003  2018-05-24 21:49:56      3.5  21.0   NaN  NaN  23.0  NaN  14.8
        2018-05-25 00:09:56      NaN   NaN   NaN  5.0   NaN  NaN   NaN
        2018-05-25 00:24:56      NaN   NaN   NaN  NaN   NaN  NaN   NaN
        2018-05-25 01:09:56      NaN   NaN   NaN  NaN   NaN  NaN   NaN
        2018-05-25 01:34:56      NaN   NaN   NaN  NaN   NaN  NaN   NaN

category          heart rate  potassium  resp_rate  sodium  \
icustay_id time
200003  2018-05-24 21:49:56      NaN      3.1      NaN  140.0
        2018-05-25 00:09:56    119.0      NaN      35.0   NaN
        2018-05-25 00:24:56      NaN      NaN      NaN   NaN
        2018-05-25 01:09:56    122.0      NaN      34.0   NaN
        2018-05-25 01:34:56    115.0      NaN      33.0   NaN

category          sys_BP  temperature  urine_out
icustay_id time
200003  2018-05-24 21:49:56      NaN      NaN      NaN
        2018-05-25 00:09:56    91.0      NaN      NaN
        2018-05-25 00:24:56      NaN      NaN     230.0
        2018-05-25 01:09:56    81.0     39.0      NaN
        2018-05-25 01:34:56    98.0      NaN      NaN
```

Recueillir les données cibles, y compris l'étiquette et le score SAPS-II L'étiquette et le score SAPS-II sont stockés dans les fichiers patient\_details et sapsii\_score. Nous chargeons les données de ces deux fichiers, puis les combinons ensemble.

```
In [18]: # charger les détails du patient à partir du fichier
path = get_path('patient_details', 'csv')
patient_df = pd.read_csv(path, index_col=0)
patient_df = patient_df.set_index('icustay_id').sort_index()
patient_df.head()
```

```
Out[18]:          subject_id  hadm_id  gender  dbsource  age  \
icustay_id
200003          27513    163557      M    carevue  48.30
```

200007	20707	129310	M	carevue	43.35
200014	9514	127229	M	carevue	84.73
200019	21789	112486	F	carevue	82.88
200021	61691	109307	M	metavision	60.85

		intime	los	survival	in_hospital_death	\
icustay_id						
200003	2199-08-02	19:50:04	19.97	75.17		0
200007	2109-02-17	10:03:37	3.24	-1.00		0
200014	2105-02-16	23:16:48	4.60	10.03		0
200019	2178-07-08	09:03:12	2.90	2.62		1
200021	2114-12-26	19:45:12	1.95	-1.00		0

	height_avg	weight_avg
icustay_id		
200003	NaN	46.4
200007	153.7	79.9
200014	172.7	96.5
200019	NaN	72.9
200021	NaN	63.0

```
In [19]: # charge le score patient SAPS-II
path = get_path('sapsii_score', 'csv')
sapsii_df = pd.read_csv(path, index_col=0)
sapsii_df = sapsii_df.set_index('icustay_id').sort_index()

# convertit la probabilité en étiquetage
sapsii_df['sapsii_prediction'] = np.where(sapsii_df['sapsii_prob']>0.5, 1, 0)
sapsii_df.head()
```

```
Out[19]:
```

	sapsii_prob	sapsii_prediction
icustay_id		
200003	0.106398	0
200007	0.029295	0
200014	0.305597	0
200019	0.552904	1
200021	0.326364	0

```
In [20]: # Créer un dataframe
feature_index = np.sort(feature_df.index.get_level_values('icustay_id').unique().values)
target_df = pd.DataFrame(index = feature_index)
target_df.index.name = 'icustay_id'

# Fusionner le patient_df et sapsii_df ensemble
target_df = pd.concat([target_df, patient_df['in_hospital_death'], sapsii_df], axis = 1)
target_df.head()
```

```
Out[20]:
```

	in_hospital_death	sapsii_prob	sapsii_prediction
icustay_id			



200003	0	0.106398	0
200007	0	0.029295	0
200014	0	0.305597	0
200019	1	0.552904	1
200021	0	0.326364	0

Nettoyage de données Copiez l'âge du patient dans la fonction Nous copions l'âge de patient\_df à feature\_df. Comme la durée des séjours en USI est courte par rapport à l'âge, on peut supposer que l'âge est une constante dans la série chronologique.

```
In [21]: feature_df = pd.merge(feature_df.reset_index(), patient_df['age'].reset_index(),
                                on="icustay_id").set_index(['icustay_id', 'time'])
```

```
In [22]: feature_df.head()
```

```
Out[22]:
```

			BILIRUBIN	BUN	FI02	GCS	HC03	P02	WBC \
icustay_id	time								
200003	2018-05-24 21:49:56		3.5	21.0	NaN	NaN	23.0	NaN	14.8
	2018-05-25 00:09:56		NaN	NaN	NaN	5.0	NaN	NaN	NaN
	2018-05-25 00:24:56		NaN	NaN	NaN	NaN	NaN	NaN	NaN
	2018-05-25 01:09:56		NaN	NaN	NaN	NaN	NaN	NaN	NaN
	2018-05-25 01:34:56		NaN	NaN	NaN	NaN	NaN	NaN	NaN

			heart rate	potassium	resp_rate	sodium \
icustay_id	time					
200003	2018-05-24 21:49:56		NaN	3.1	NaN	140.0
	2018-05-25 00:09:56		119.0	NaN	35.0	NaN
	2018-05-25 00:24:56		NaN	NaN	NaN	NaN
	2018-05-25 01:09:56		122.0	NaN	34.0	NaN
	2018-05-25 01:34:56		115.0	NaN	33.0	NaN

			sys_BP	temperature	urine_out	age
icustay_id	time					
200003	2018-05-24 21:49:56		NaN	NaN	NaN	48.3
	2018-05-25 00:09:56		91.0	NaN	NaN	48.3
	2018-05-25 00:24:56		NaN	NaN	230.0	48.3
	2018-05-25 01:09:56		81.0	39.0	NaN	48.3
	2018-05-25 01:34:56		98.0	NaN	NaN	48.3

Détecter les valeurs aberrantes Nous obtenons d'abord une brève statistique de la base de données de la fonctionnalité.

```
In [23]: feature_df.describe()
```

```
Out[23]:
```

	BILIRUBIN	BUN	FI02	GCS	HC03 \
count	17413.000000	61019.000000	9781.000000	195545.000000	58402.000000
mean	2.122776	26.763787	50.414641	4.047260	23.237726
std	4.583922	22.705096	14.479749	1.268771	4.870280
min	0.000000	1.000000	0.300000	1.000000	2.000000

25%	0.400000	13.000000	40.000000	3.330000	21.000000
50%	0.700000	19.000000	50.000000	4.670000	23.000000
75%	1.700000	32.000000	60.000000	5.000000	26.000000
max	82.800000	272.000000	99.000000	6.000000	53.000000

	P02	WBC	heart rate	potassium \
count	71477.000000	57305.000000	727503.000000	100003.000000
mean	183.363753	12.365824	85.510838	4.226053
std	118.540812	10.725286	18.618176	0.786722
min	10.000000	0.100000	0.350000	0.600000
25%	94.000000	7.800000	72.000000	3.700000
50%	143.000000	10.800000	84.000000	4.100000
75%	251.000000	14.800000	97.000000	4.600000
max	689.000000	600.200000	222.000000	17.500000

	resp_rate	sodium	sys_BP	temperature \
count	725291.000000	77621.000000	678309.000000	219675.000000
mean	18.841348	138.187536	119.567359	36.896975
std	5.542793	5.591240	22.951818	0.863139
min	1.000000	1.210000	0.350000	20.900000
25%	15.000000	136.000000	103.000000	36.390000
50%	18.000000	138.000000	117.000000	36.890000
75%	22.000000	141.000000	134.000000	37.440000
max	69.000000	182.000000	323.000000	42.780000

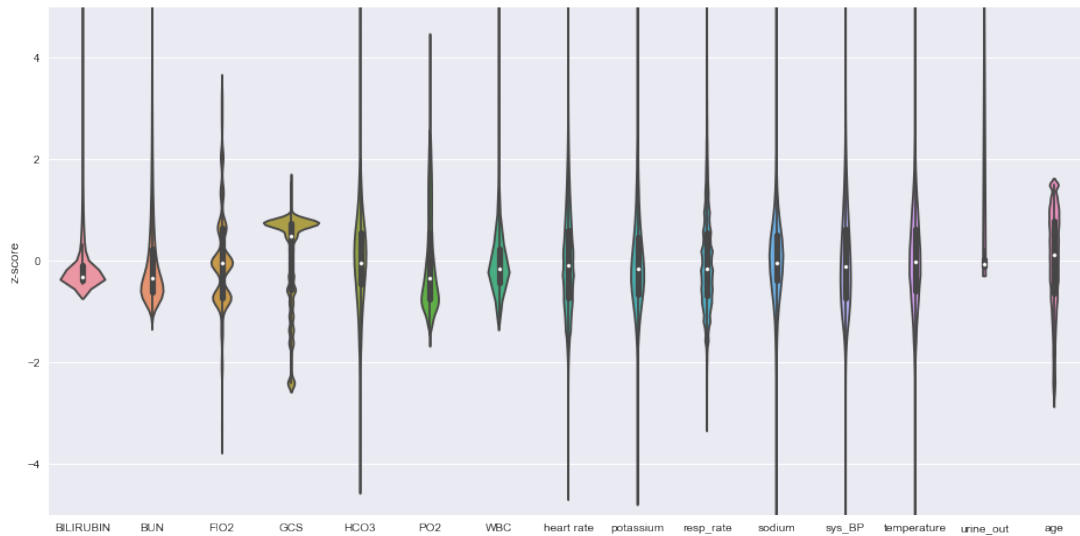
	urine_out	age
count	355367.000000	1.013695e+06
mean	138.546588	6.470788e+01
std	949.048086	1.774936e+01
min	0.300000	1.602000e+01
25%	40.000000	5.335000e+01
50%	80.000000	6.677000e+01
75%	170.000000	7.884000e+01
max	555975.000000	9.140000e+01

Basé sur les statistiques, certaines mesures sont loin de la distribution et pourraient être aberrantes. Nous utilisons ensuite le score z pour déterminer les valeurs aberrantes.

```
In [24]: # Calcule le z-score des données
zscore = (feature_df - feature_df.mean()) / feature_df.std()

# Boxplot du z-score
boxplot_dims = (16, 8)
fig, ax = plt.subplots(figsize=boxplot_dims)
sns.violinplot(ax=ax, data=zscore)
ax.set_ylabel('z-score')
ax.set_ylim([-5,5])
```

```
Out[24]: (-5, 5)
```



GCS est compris entre 1 et 5, les données hors de cette plage sont considérées comme des valeurs aberrantes. Pour la variable *âge*, la fourchette devrait être comprise entre 16 et 91,4 (l'âge maximum). Pour le reste des caractéristiques, nous considérons les observations comme aberrantes si leur z-score est supérieur ou inférieur à 3,5 ou -3,5. Ensuite, nous remplaçons la valeur des valeurs aberrantes par NaN.

```
In [26]: # Conditions des valeurs aberrantes
for feature in (set(feature_df.columns) - set(['GCS', 'age'])):
    feature_df.loc[np.abs(zscore[feature]) > 3.5, feature] = np.nan

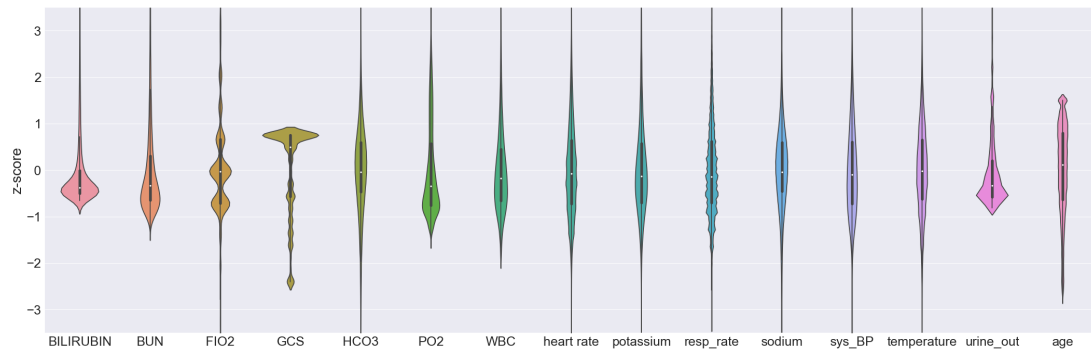
# GCS moyen devrait être entre le plus bas (3/3 = 1) au plus haut (15/3 = 5)
feature_df.loc[(feature_df['GCS'] > 5) | (feature_df['GCS'] < 1), 'GCS'] = np.nan

# La tranche d'âge se situe entre 16 et 91,4
feature_df.loc[(feature_df['age'] > 91.4) | (feature_df['age'] < 16), 'age'] = np.nan

In [27]: # Calcule le score z des données
zscore = (feature_df - feature_df.mean()) / feature_df.std()

# Boxplot du score z
boxplot_dims = (30, 10)
sns.set(font_scale=2.5)
fig, ax = plt.subplots(figsize=boxplot_dims)
sns.violinplot(ax=ax, data=zscore)
ax.set_ylabel('z-score')
ax.set_ylim([-3.5, 3.5])
if not os.path.exists('plot'):
    os.makedirs('plot')
plt.tight_layout()
plt.savefig('plot/distribution_of_z-score')
```

```
plt.show()
plt.close()
```



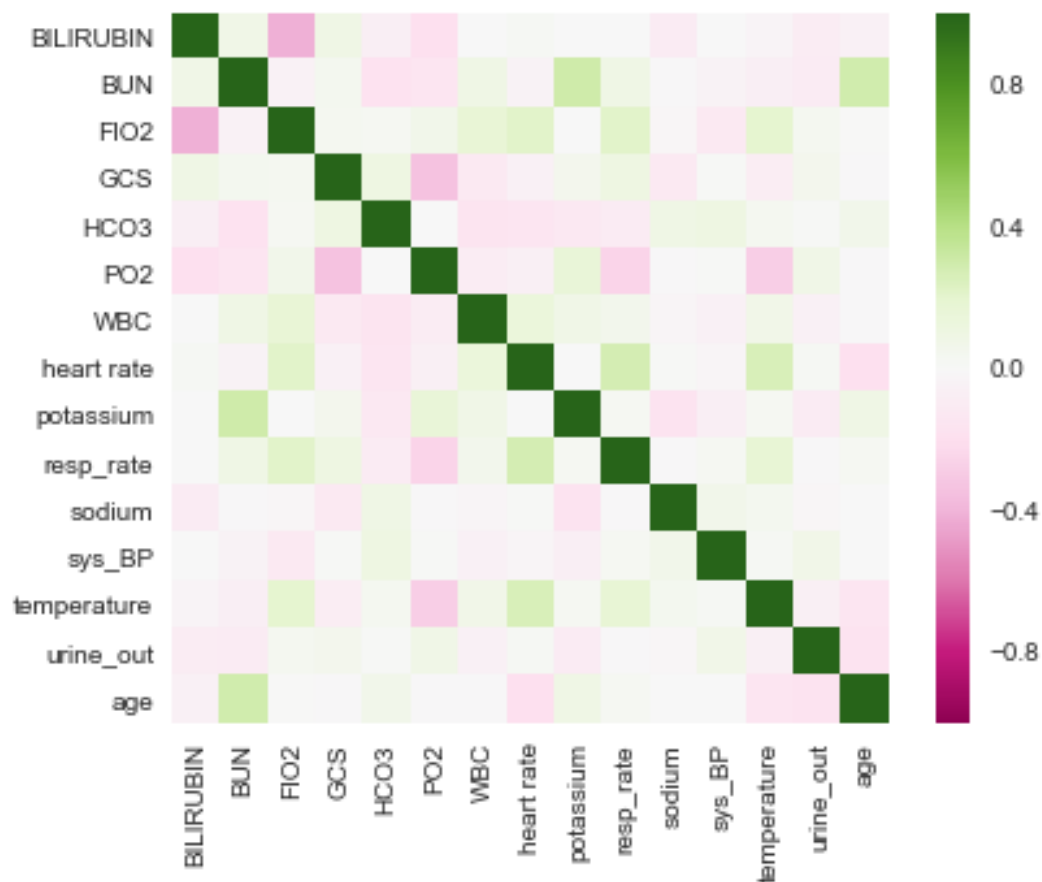
Corrélation de fonctionnalité Ensuite, nous aimerions la corrélation entre chaque fonctionnalité.

```
In [25]: # calcule la matrice de corrélation
corr_matrix = feature_df.corr()

# tracer la carte de chaleur
boxplot_dims = (6, 5)
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=boxplot_dims)
sns.heatmap(ax=ax, data=corr_matrix,
            xticklabels=corr_matrix.columns,
            yticklabels=corr_matrix.columns,
            cmap='PiYG',
            vmin = -1,
            vmax = 1)

if not os.path.exists('plot'):
    os.makedirs('plot')
plt.tight_layout()
plt.savefig('plot/corr_matrix')
plt.show()
plt.close()

corr_matrix
```



```

Out[25]:
      BILIRUBIN      BUN      FIO2      GCS      HCO3      PO2 \
BILIRUBIN      1.000000      0.084949     -0.417520      0.093124     -0.080506     -0.202875
BUN              0.084949      1.000000     -0.048806      0.041355     -0.186975     -0.152698
FIO2             -0.417520     -0.048806      1.000000      0.033098      0.024963      0.066359
GCS              0.093124      0.041355      0.033098      1.000000      0.104613     -0.336717
HCO3             -0.080506     -0.186975      0.024963      0.104613      1.000000     -0.004569
PO2             -0.202875     -0.152698      0.066359     -0.336717     -0.004569      1.000000
WBC             -0.004729      0.087703      0.168559     -0.123843     -0.165576     -0.094996
heart rate       0.017654     -0.039634      0.217693     -0.058132     -0.159880     -0.068285
potassium       -0.000558      0.300214     -0.000389      0.047847     -0.132456      0.159274
resp_rate       -0.003573      0.087007      0.216232      0.108720     -0.106846     -0.251019
sodium          -0.107307     -0.012524     -0.022445     -0.120784      0.090062     -0.012576
sys_BP          -0.005067     -0.045894     -0.118602      0.014297      0.107411      0.013034
temperature     -0.032358     -0.074448      0.200823     -0.089134      0.034297     -0.281525
urine_out       -0.100774     -0.103864      0.033038      0.047995      0.007984      0.084234
age             -0.060121      0.291713      0.000894     -0.013157      0.068086     -0.008825

      WBC      heart rate      potassium      resp_rate      sodium      sys_BP \

```

BILIRUBIN	-0.004729	0.017654	-0.000558	-0.003573	-0.107307	-0.005067
BUN	0.087703	-0.039634	0.300214	0.087007	-0.012524	-0.045894
FI02	0.168559	0.217693	-0.000389	0.216232	-0.022445	-0.118602
GCS	-0.123843	-0.058132	0.047847	0.108720	-0.120784	0.014297
HCO3	-0.165576	-0.159880	-0.132456	-0.106846	0.090062	0.107411
PO2	-0.094996	-0.068285	0.159274	-0.251019	-0.012576	0.013034
WBC	1.000000	0.143979	0.085844	0.059692	-0.031238	-0.056317
heart_rate	0.143979	1.000000	-0.003631	0.278479	0.009188	-0.030544
potassium	0.085844	-0.003631	1.000000	0.025003	-0.172057	-0.077325
resp_rate	0.059692	0.278479	0.025003	1.000000	-0.008487	0.024290
sodium	-0.031238	0.009188	-0.172057	-0.008487	1.000000	0.069074
sys_BP	-0.056317	-0.030544	-0.077325	0.024290	0.069074	1.000000
temperature	0.076042	0.264115	0.024961	0.175112	0.045899	0.016717
urine_out	-0.055682	0.015773	-0.104999	-0.008590	-0.017163	0.072786
age	-0.010598	-0.201844	0.093242	0.023649	-0.007173	-0.002676

	temperature	urine_out	age
BILIRUBIN	-0.032358	-0.100774	-0.060121
BUN	-0.074448	-0.103864	0.291713
FI02	0.200823	0.033038	0.000894
GCS	-0.089134	0.047995	-0.013157
HCO3	0.034297	0.007984	0.068086
PO2	-0.281525	0.084234	-0.008825
WBC	0.076042	-0.055682	-0.010598
heart_rate	0.264115	0.015773	-0.201844
potassium	0.024961	-0.104999	0.093242
resp_rate	0.175112	-0.008590	0.023649
sodium	0.045899	-0.017163	-0.007173
sys_BP	0.016717	0.072786	-0.002676
temperature	1.000000	-0.070302	-0.159375
urine_out	-0.070302	1.000000	-0.176012
age	-0.159375	-0.176012	1.000000

Rééchantillonnage, interpolation et imputation des données Nous rééchantillons les séries chronologiques irrégulières en séries temporelles régulières avec un intervalle de temps de 2 heures. Ensuite, nous interpolons la série chronologique par chaque patient. Enfin, nous imputons l'utilisation de la moyenne si toute la série temporelle est manquante

In [28]: # Rééchantillonner l'index temporel

```
time_index = pd.date_range(datetime.date.today(), periods=12, freq='2h', name='time')
resampled_df = feature_df.groupby(level=0).apply(lambda group: group.resample('2h', level=0).bfill())
resampled_df = resampled_df.groupby(level=0).apply(lambda group: group.reset_index(level=0))
```

In [32]: # créer une étiquette

```
labels = ['bilirubin', 'blood urea nitrogen', 'fraction of inspired oxygen', 'glasgow coma scale',
          'partial pressure of oxygen', 'white blood cell', 'heart rate', 'potassium',
          'sodium', 'systolic blood pressure', 'temperature', 'urine output volume', 'age']
```

```

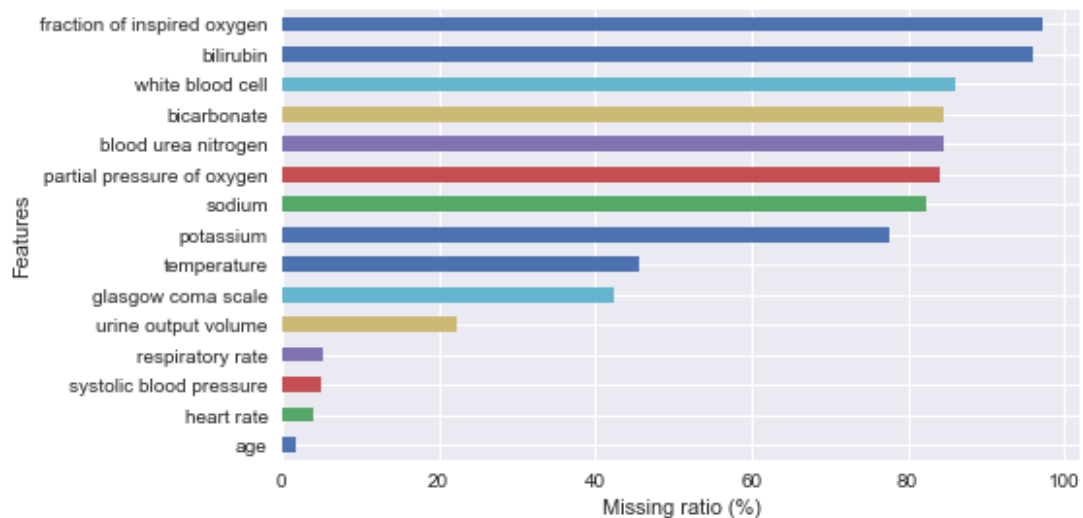
# calcule le taux manquant
missing_ratio = (100*(1 - resampled_df.count()/resampled_df.shape[0]))

# remplacer l'étiquette
missing_ratio.index = labels

# trier par valeurs
missing_ratio = missing_ratio.sort_values(ascending=True)

# créer une parcelle pour voir le ratio manquant
plot_dims = (8, 4)
fig, ax = plt.subplots(figsize=plot_dims)
missing_ratio.plot(ax=ax, kind = 'barh')
ax.set_ylabel('Features')
ax.set_xlabel('Missing ratio (%)')
if not os.path.exists('plot'):
    os.makedirs('plot')
plt.tight_layout()
plt.savefig('plot/missing_ratio')
plt.show()
plt.close()

```



In [33]: resampled\_df.count()

```

Out[33]: BILIRUBIN      11470
         BUN            44356
         FIO2           8308
         GCS           165978
         HCO3           44229

```

```

P02          46203
WBC          40430
heart_rate   276461
potassium    64205
resp_rate    273042
sodium       50998
sys_BP       273612
temperature  156495
urine_out    224168
age          283368
dtype: int64

```

```
In [37]: resampled_df.head()
```

```

Out[37]:
          BILIRUBIN  BUN  FIO2  GCS  HCO3  P02  WBC  \
icustay_id time
200003  2018-05-25 00:00:00    NaN  NaN  NaN  5.0  NaN  NaN
        2018-05-25 02:00:00    NaN  NaN  NaN  NaN  NaN  NaN
        2018-05-25 04:00:00    3.4  20.0  NaN  5.0  18.0  NaN  40.2
        2018-05-25 06:00:00    NaN  NaN  NaN  NaN  NaN  91.0  NaN
        2018-05-25 08:00:00    NaN  NaN  NaN  NaN  NaN  NaN  NaN

          heart_rate  potassium  resp_rate  sodium  \
icustay_id time
200003  2018-05-25 00:00:00  118.200000    NaN    32.60    NaN
        2018-05-25 02:00:00  114.400000    NaN    31.20    NaN
        2018-05-25 04:00:00  112.166667    3.2    33.00  141.0
        2018-05-25 06:00:00  109.000000    NaN    33.25    NaN
        2018-05-25 08:00:00   93.000000    NaN    37.00    NaN

          sys_BP  temperature  urine_out  age
icustay_id time
200003  2018-05-25 00:00:00   89.000000    39.00   230.00  48.3
        2018-05-25 02:00:00   85.800000    38.03    90.00  48.3
        2018-05-25 04:00:00   96.333333    36.83   128.50  48.3
        2018-05-25 06:00:00  104.800000    36.44   263.75  48.3
        2018-05-25 08:00:00  112.500000    NaN   300.00  48.3

```

Les données manquantes sont traitées par imputation selon les étapes suivantes: appliquer une interpolation linéaire sur la série chronologique multivariée pour chaque patient remplir les données manquantes en utilisant la prochaine observation valide pour chaque patient Certaines observations sont toujours manquantes après ces imputations, car il y a des données manquantes pour certaines fonctions chez certains patients. Dans ce cas, nous remplissons la série chronologique en utilisant la moyenne de toutes les observations.

```
In [38]: from sklearn.preprocessing import Imputer, MinMaxScaler
```

```

# Interpoler
resampled_df = resampled_df.groupby(level=0).apply(lambda group: group.interpolate())

```



```

resampled_df = resampled_df.groupby(level=0).apply(lambda group: group.fillna(method='b

# Imputate utilisant la moyenne si toute la série temporelle est manquante
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imputed_data = imp.fit_transform(resampled_df)

# mettre les données à l'échelle de [0,1]
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(imputed_data)

# Stocker l'image traitée en tant que processing_df
processed_feature_df = pd.DataFrame(scaled_data, index=resampled_df.index, columns=resa

```

In [39]: processed\_feature\_df.head()

```

Out[39]:

```

		BILIRUBIN	BUN	FI02	GCS	HC03	\
icustay_id	time						
200003	2018-05-25 00:00:00	0.187845	0.180952	0.499414	1.0	0.333333	
	2018-05-25 02:00:00	0.187845	0.180952	0.499414	1.0	0.333333	
	2018-05-25 04:00:00	0.187845	0.180952	0.499414	1.0	0.333333	
	2018-05-25 06:00:00	0.191298	0.179762	0.499414	1.0	0.359848	
	2018-05-25 08:00:00	0.194751	0.178571	0.499414	1.0	0.386364	

		P02	WBC	heart rate	potassium	\
icustay_id	time					
200003	2018-05-25 00:00:00	0.132075	0.805221	0.749606	0.314815	
	2018-05-25 02:00:00	0.132075	0.805221	0.719685	0.314815	
	2018-05-25 04:00:00	0.132075	0.805221	0.702100	0.314815	
	2018-05-25 06:00:00	0.132075	0.814508	0.677165	0.312500	
	2018-05-25 08:00:00	0.139365	0.823795	0.551181	0.310185	

		resp_rate	sodium	sys_BP	temperature	\
icustay_id	time					
200003	2018-05-25 00:00:00	0.854054	0.578947	0.308176	0.850250	
	2018-05-25 02:00:00	0.816216	0.578947	0.288050	0.688852	
	2018-05-25 04:00:00	0.864865	0.578947	0.354298	0.489185	
	2018-05-25 06:00:00	0.871622	0.588816	0.407547	0.424293	
	2018-05-25 08:00:00	0.972973	0.598684	0.455975	0.420133	

		urine_out	age
icustay_id	time		
200003	2018-05-25 00:00:00	0.067565	0.42823
	2018-05-25 02:00:00	0.026385	0.42823
	2018-05-25 04:00:00	0.037709	0.42823
	2018-05-25 06:00:00	0.077492	0.42823
	2018-05-25 08:00:00	0.088155	0.42823

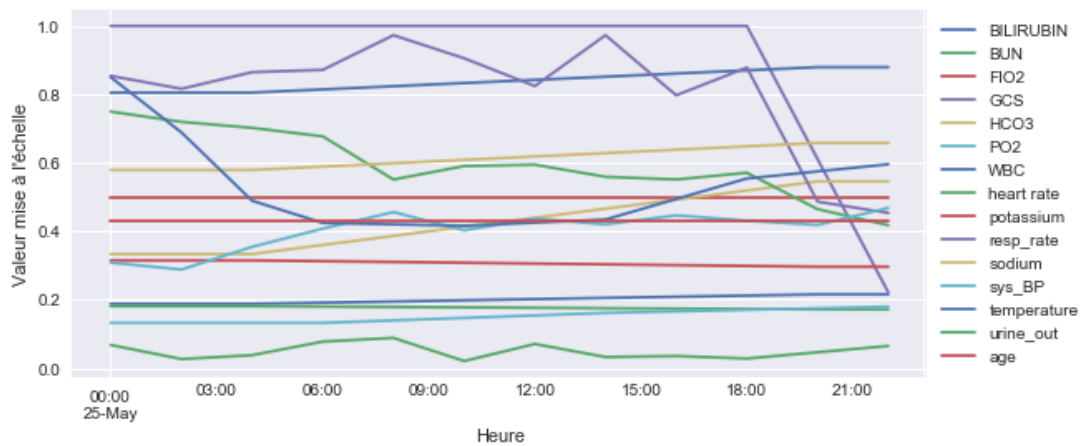
Nous illustrons un exemple des données d'origine et des données traitées.

In [31]: *# tracer la série chronologique pour un patient*

```

plot_dims = (8, 4)
fig, ax = plt.subplots(figsize=plot_dims)
processed_feature_df.loc[(200003), :].plot(ax=ax)
ax.set_ylabel("Valeur mise à l'échelle")
ax.set_xlabel('Heure')
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
if not os.path.exists('plot'):
    os.makedirs('plot')
plt.tight_layout()
plt.savefig('plot/An_example_of_imputed_and_transformed_data')
plt.show()
plt.close()

```



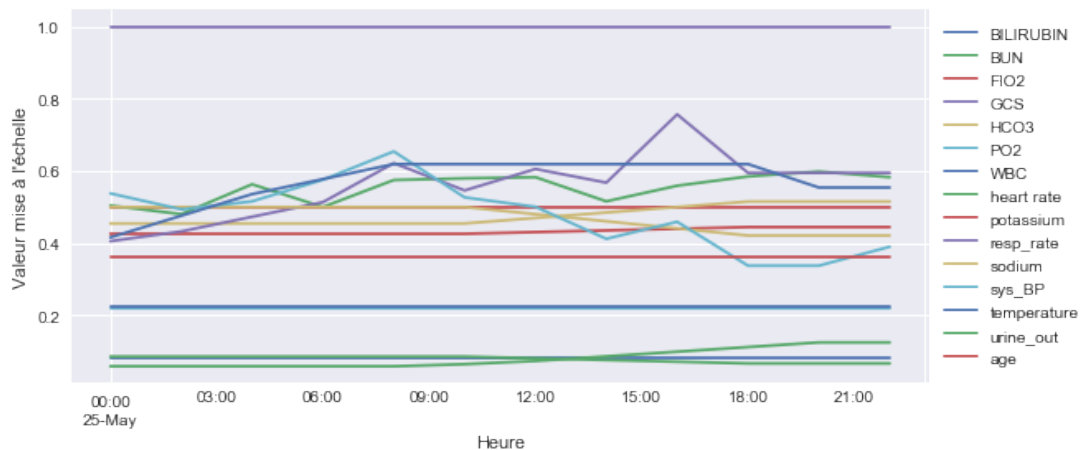
In [40]: *# tracer la série chronologique pour un autre patient*

```

plot_dims = (8, 4)
fig, ax = plt.subplots(figsize=plot_dims)

processed_feature_df.loc[(200007), :].plot(ax=ax)
ax.set_ylabel("Valeur mise à l'échelle")
ax.set_xlabel('Heure')
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
if not os.path.exists('plot'):
    os.makedirs('plot')
plt.tight_layout()
plt.savefig('plot/An_example_of_imputed_and_transformed_data')
plt.show()
plt.close()

```



Créer un train, valide et tester des données

```
In [36]: positives = (target_df['in_hospital_death'] == 1).sum()
negatives = (target_df['in_hospital_death'] == 0).sum()
sum_all = positives + negatives
positives = positives/sum_all*100
negatives = negatives/sum_all*100

print('Le pourcentage de positifs et de négatifs: ', positives, negatives)
```

Le pourcentage de positifs et de négatifs: 11.441771414301174 88.55822858569883

L'ensemble de données est déséquilibré et contient environ 11,4% d'échantillons positifs. Afin de préserver le pourcentage d'échantillons pour chaque classe, une répartition aléatoire stratifiée est appliquée à l'ensemble de données. La taille de la taille de test est définie sur 20% de l'ensemble de données d'origine, qui contient 4806 échantillons. D'un autre côté, l'ensemble de données d'apprentissage contient 19220 échantillons. Enfin, les ensembles de données d'entraînement et de test sont divisés en 'X\_train', 'X\_test', 'y\_train' et 'y\_test'.

```
In [38]: # Importer train_test_split
#from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit

# Transformer les données en liste de données
features = []
targets = []

for icustay_id, sub_df in processed_feature_df.groupby(level=0):
    features.append(processed_feature_df.loc[icustay_id,:].values)
    targets.append(target_df.loc[icustay_id, :].values)
```

```

# considérer seulement l'étiquette
target_vector = np.array(targets)[: ,0]

# Appliquer stratifiedShuffleSplit pour conserver l'ensemble déséquilibré
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=0)
indexs = next(iter(sss.split(features, target_vector)))

X_train = [features[index] for index in indexs[0]]
y_train = [targets[index] for index in indexs[0]]
X_test = [features[index] for index in indexs[1]]
y_test = [targets[index] for index in indexs[1]]

# Afficher les résultats de la scission
print("L'ensemble d'entraînement a {} échantillons.".format(len(X_train)))
print("L'ensemble de test a {} échantillons.".format(len(X_test)))

# Check the folds are made by preserving the percentage of samples for each class
difference = abs(sum(np.array(y_train)[: ,0])/len(y_train) - sum(np.array(y_test)[: ,0])/len(y_test))
print("La différence de la fraction d'échantillons positifs entre le train et les jeux de test est {}".format(difference))

```

L'ensemble d'entraînement a 19220 échantillons.

L'ensemble de test a 4806 échantillons.

La différence de la fraction d'échantillons positifs entre le train et les jeux de données de test est 0.0001.

Les données de caractéristiques sont une liste de tableaux, et il est préférable de convertir les données qui peuvent être stockées en utilisant le format csv. En outre, supprimez l'ID d'origine ICU-stay et l'intervalle de temps.

```

In [40]: # créer un index contient l'identifiant et l'heure pour les données d'apprentissage
time_index = np.arange(0,12,1)
id_index = np.arange(0, len(X_train), 1)
combined_index = pd.MultiIndex.from_product([id_index, time_index], names=['id', 'time'])

# Convertir les données d'apprentissage en données
X_train = pd.DataFrame(np.concatenate(X_train, axis=0), index=combined_index, columns=target_df.columns)
y_train = pd.DataFrame(np.array(y_train), index=id_index, columns=target_df.columns)
y_train.index.name = 'id'

In [41]: # Convertir les données d'apprentissage en données
id_index = np.arange(len(y_train), len(y_train) + len(y_test), 1)
combined_index = pd.MultiIndex.from_product([id_index, time_index], names=['id', 'time'])

# convertir les données de test en données
X_test = pd.DataFrame(np.concatenate(X_test, axis=0), index=combined_index, columns=target_df.columns)
y_test = pd.DataFrame(np.array(y_test), index=id_index, columns=target_df.columns)
y_test.index.name = 'id'

```

```
In [42]: for filename, data in zip(['X_train', 'X_test', 'y_train', 'y_test'],
                                   [X_train, X_test, y_train, y_test]):
        data.to_csv(get_path(filename, 'csv'))
```

```
In [43]: X_train.head()
```

```
Out[43]:
```

		BILIRUBIN	BUN	FI02	GCS	HC03	P02	WBC	\
	id time								
0	0	0.082443	0.171429	0.499414	1.0	0.484848	0.219736	0.068273	
	1	0.082443	0.171429	0.499414	1.0	0.484848	0.219736	0.068273	
	2	0.082443	0.171429	0.499414	1.0	0.515152	0.219736	0.074297	
	3	0.082443	0.171429	0.499414	1.0	0.545455	0.219736	0.080321	
	4	0.082443	0.171429	0.499414	1.0	0.575758	0.219736	0.086345	

		heart rate	potassium	resp_rate	sodium	sys_BP	temperature	\
	id time							
0	0	0.307087	0.222222	0.513514	0.631579	0.723270	0.564060	
	1	0.232283	0.222222	0.432432	0.631579	0.691824	0.512895	
	2	0.225722	0.305556	0.445946	0.605263	0.603774	0.461730	
	3	0.267717	0.388889	0.500000	0.578947	0.679245	0.397671	
	4	0.275591	0.472222	0.581081	0.552632	0.676101	0.401830	

		urine_out	age
	id time		
0	0	0.014619	1.0
	1	0.014619	1.0
	2	0.058740	1.0
	3	0.026385	1.0
	4	0.073448	1.0

```
In [44]: X_test.head()
```

```
Out[44]:
```

		BILIRUBIN	BUN	FI02	GCS	HC03	P02	WBC	\
	id time								
19220	0	0.082443	0.104762	0.499414	1.0	0.545455	0.219736	0.251004	
	1	0.082443	0.104762	0.499414	1.0	0.545455	0.219736	0.251004	
	2	0.082443	0.104762	0.499414	1.0	0.545455	0.219736	0.251004	
	3	0.082443	0.104762	0.499414	1.0	0.545455	0.219736	0.251004	
	4	0.082443	0.104762	0.499414	1.0	0.545455	0.219736	0.251004	

		heart rate	potassium	resp_rate	sodium	sys_BP	temperature	\
	id time							
19220	0	0.637795	0.407407	0.567568	0.447368	0.537736	0.720466	
	1	0.669291	0.407407	0.432432	0.447368	0.591195	0.720466	
	2	0.606299	0.407407	0.432432	0.447368	0.522013	0.720466	
	3	0.614173	0.407407	0.513514	0.447368	0.515723	0.697171	
	4	0.511811	0.407407	0.459459	0.447368	0.459119	0.673877	

		urine_out	age
	id time		
19220	0		
	1		
	2		
	3		
	4		

id	time		
19220	0	0.035209	0.050279
	1	0.029326	0.050279
	2	0.052858	0.050279
	3	0.041092	0.050279
	4	0.017560	0.050279

In [45]: `y_train.head()`

	in_hospital_death	sapsii_prob	sapsii_prediction
id			
0	0.0	0.096698	0.0
1	0.0	0.391926	0.0
2	0.0	0.140051	0.0
3	0.0	0.096698	0.0
4	0.0	0.152870	0.0

In [46]: `y_test.head()`

	in_hospital_death	sapsii_prob	sapsii_prediction
id			
19220	0.0	0.004584	0.0
19221	0.0	0.079390	0.0
19222	1.0	0.166523	0.0
19223	0.0	0.052195	0.0
19224	0.0	0.305597	0.0

Annexe E

Data-Analyse

# analyse-data

May 29, 2018

L'analyse des données Le but de ce cahier est de prédire la mort à l'hôpital en visite à l'USI sur la base des 24 premières heures de séjour à l'unité de soins intensifs.

```
In [2]: # Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial import distance
import seaborn as sns
import urllib.request
import zipfile
import os
from time import time

# custom py files
import dtw
import KNN
import visuals as vs

# for pretty printing pandas dataframes
from IPython.display import display, HTML

%matplotlib inline
plt.style.use('seaborn-notebook')
```

Télécharger les données

```
In [3]: # generate a data folder, we store all data in this folder
if not os.path.exists('data'):
    os.makedirs('data')

# download and unzip the rawdata
url = 'https://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly8xZHJ2Lm1zL3UvcyFBdk96S3k0djAydH
urllib.request.urlretrieve(url, "download.zip")

with zipfile.ZipFile("download.zip","r") as zip_ref:
    zip_ref.extractall("data")
```



Les fonctions

```
In [4]: def get_path(filename, suffix):
        """
        Return the path of the file.

        inputs:
        - filename: string that points the file
        - suffix: string that indicates the file format
        """
        cwd = os.getcwd()
        dir_name = os.path.join(cwd, 'data')
        return os.path.join(dir_name, filename + "." + suffix)
```

Charger les données Ce sont des données de train, validation et de test, et toutes sont chargées dans un dictionnaire nommé 'data'. Le multi-index représente deux dimensions, l'id du patient et le temps.

```
In [6]: names = ['X_train', 'X_test', 'y_train', 'y_test']

        data = {(name):(pd.read_csv(get_path(name, 'csv'), index_col = [0,1]) if name.split('_')
                                else pd.read_csv(get_path(name, 'csv'), index_col = 0)) for name in names}
```

Données DTW Ensuite, nous générons les données pour le classificateur kNN de distorsion temporelle dynamique (DTW).

```
In [10]: # data for DTW kNN classifier
        data_dtw = {}

        for X in ['X_test', 'X_train']:
            df = data[X]
            data_dtw[X] = []
            for key, value in df.groupby(level=0):
                data_dtw[X].append(value.values)
            data_dtw[X] = np.array(data_dtw[X])
```

Données pour les classificateurs conventionnels En plus d'utiliser le DTN kNN, nous utilisons également des classificateurs conventionnels pour prédire la mort à l'hôpital. Mais, nous devons d'abord extraire de nouvelles fonctionnalités de la série temporelle multivariée et réduire les dimensions des données de la 3D à la 2D.

```
In [11]: # données pour les classificateurs conventionnels
        data_agg = {}

        # extraire les propriétés statistiques telles que moyenne, médiane, max, min, std pour
        for name, df in data.items():
            data_agg[name] = df.groupby('id').agg([np.median, max, min, np.mean, np.std])
            data_agg[name].columns = ['_'.join( ( col[0], col[1])) for col in data_agg[name].columns]
```

En outre, nous nous intéressons également au nombre de points de données au-dessous ou au-dessus des Q1 et Q3 globaux pour chaque patient.

```

In [12]: # Fonction pour compter les points de données au-dessous / au-dessus du seuil
def count_below_threshold(array, threshold):
    '''Compter les points de données inférieurs au seuil dans la série chronologique'''
    return np.sum(array < threshold.reshape(1,array.shape[1]), axis=0)/array.shape[0]

def count_above_threshold(array, threshold):
    '''Compter les points de données au-dessus du seuil dans la série chronologique'''
    return np.sum(array > threshold.reshape(1,array.shape[1]), axis=0)/array.shape[0]

data_quartiles = {}

# constantes
count_funcs = [count_below_threshold, count_above_threshold]
column_names = ['below_Q1', 'above_Q3']
quartiles = {key: data["X_train"].quantile(value) \
              for (key,value) in zip(['Q1', 'Q3'], [0.25, 0.75])} # global quantiles of

# Boucle pour créer de nouvelles fonctionnalités et les stocker dans des quartiles de d
for name, df in data.items():
    if name.split('_')[0] == 'X': # Only consider the training data
        data_quartiles[name] = {}
        for Q, func, column in zip(['Q1', 'Q3'], count_funcs, column_names):
            data_quartiles[name][Q] = df.groupby('id').apply(lambda x: func(x, quartile
            data_quartiles[name][Q].columns = ['_'.join( ( col, column)) for col in dat

# Concat data_quantile into data_agg
for name in ['X_train', 'X_test']:
    combined_df = pd.concat( [data_quartiles[name]['Q1'], data_quartiles[name]['Q3'], d
    data_agg[name] = combined_df

```

Calcul des scores de Benchmark

```

In [13]: from sklearn.metrics import f1_score, matthews_corrcoef

# stocker les partitions dans un dictionnaire
score_metrics = {'f1': f1_score, 'mcc': matthews_corrcoef}

# crée le score pour Benchmark
score_benchmark = {}

# performance de Benchmark
for name, func in score_metrics.items():
    score_benchmark[name+'_test'] = func(data['y_test']['in_hospital_death'], data['y_t
    score_benchmark[name+'_train'] = func(data['y_train']['in_hospital_death'], data['y

print("Le score F1 et le MCC du benchmark des données de test sont: {}, {}".format(score_
score_

# performance de naive learner

```

```

score_naive = {}
# sur le sous-ensemble d'échantillons de test
score_naive['f1_test'] = f1_score(data['y_test']['in_hospital_death'][:480], np.ones(
score_naive['mcc_test'] = matthews_corrcoef(data['y_test']['in_hospital_death'][:480],

print("Le score F1 et le MCC du prédicteur naïf sont: {}, {}".format(score_naive['f1_te

```

Le score F1 et le MCC du benchmark des données de test sont: 0.3713768115942029, 0.2898101625079  
Le score F1 et le MCC du prédicteur naïf sont: 0.2189239332096475, 0.0

```

/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:538: RuntimeWarning: in
mcc = cov_ytyp / np.sqrt(cov_ytyp * cov_ypyp)

```

Prédiction avec le classificateur DTN kNN Ici, c'est la première tentative de former l'apprenant en DTW avec un classificateur KNN.

In [14]: results = {}

```

# Custom metric
def dtw_metric(a,b):
    similarity, _ = dtw.dynamic_time_wrapping(a,b, distance.euclidean)
    return similarity

# initize the classifier
dtw_kNN_clf = KNN.KNNClassifier(metric=dtw_metric)

# First, we start with 1922 training samples since kNN is a slow classifier
start = time() # Get start time
dtw_kNN_clf.fit(data_dtw['X_train'][:1922], data['y_train']['in_hospital_death'][:1922])
end = time() # Get end time

# Record the training time
results['train_time'] = end - start

# Make predictions on train subset (with the size of test set) and test set
start = time() # Get start time
predictions_test = dtw_kNN_clf.predict(data_dtw['X_test'][:480])
end = time() # Get end time
results['pred_time'] = end - start

# Compute F-score on the train subset and test set
for name, func in score_metrics.items():
    results[name+'_test'] = func(data['y_test']['in_hospital_death'][:480], predictions

# Return the results
results

```

```
Out[14]: {'f1_test': 0.27499999999999997,
          'mcc_test': 0.2611594868357666,
          'pred_time': 2432.8259539604187,
          'train_time': 0.0040738582611083984}
```

Les résultats suggèrent que le classificateur DTN kNN est coûteux en calcul. Ses performances ne sont pas non plus très prometteuses. Prédiction avec les classificateurs conventionnels La deuxième tentative dans ce projet consiste à utiliser des méthodes de classification habituelles telles que la régression logistique. Dans ce projet, la régression logistique, l'arbre de décision, le boost de gradient, et les classificateurs de perceptron multicouche sont utilisés.

```
In [15]: def train_predict(learner, sample_size, X_train, y_train, X_test, y_test, metrics):
        '''
            contributions:
            āāāāāā - learner: l'algorithme d'apprentissage à former et à prédire sur
            āāāāāā - sample_size: la taille des échantillons (nombre) à tirer de l'ensemble d'apprentissage
            āāāāāā - X_train: ensemble d'exercices de fonctionnalités
            āāāāāā - y_train: ensemble de formation au revenu
            āāāāāā - X_test: ensemble de tests de fonctionnalités
            āāāāāā - y_test: ensemble de tests de revenu
            āāāāāā - metrics: dictionnaire pour stocker les métriques
            āāāā '''
        results = {}

        # Ajuster l'apprenant aux données d'apprentissage en les coupant avec 'sample_size'
        start = time() # Get start time
        learner.fit(X_train[:sample_size], y_train[:sample_size])
        end = time() # Get end time

        # Enregistrer le temps de formation
        results['train_time'] = end - start

        # Faire des prédictions sur les données de train
        start = time() # Get start time
        predictions_test = learner.predict(X_test)
        predictions_train = learner.predict(X_train)
        end = time() # Get end time
        results['pred_time'] = end - start

        for score, func in metrics.items():
            # Calculer le score sur le sous-ensemble du train et l'ensemble de test
            results[score+"_train"] = func(y_train, predictions_train)
            results[score+"_test"] = func(y_test, predictions_test)

        # Succès
        print("{} formé sur {} échantillons.".format(learner.__class__.__name__, sample_size))

        # Retourner les résultats
        return results
```

## Formation initiale

```
In [33]: from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import GradientBoostingClassifier
         from keras.models import Sequential
         from keras.layers import Dense, Dropout, Flatten
         from keras.wrappers.scikit_learn import KerasClassifier

         # fixe la graine aléatoire pour la reproductibilité
         seed = 0
         np.random.seed(seed)

         # créer un wrapper keras
         def keras_model():
             # créer un modèle
             clf_MLP = Sequential()
             clf_MLP.add(Dense(128, activation='relu', input_shape= data_agg['X_train'].shape[1:]))
             clf_MLP.add(Dropout(0.1))
             clf_MLP.add(Dense(32, activation='relu'))
             clf_MLP.add(Dropout(0.1))
             clf_MLP.add(Dense(8, activation='relu'))
             clf_MLP.add(Dropout(0.1))
             clf_MLP.add(Dense(1, activation='sigmoid'))
             #clf_MLP.summary()
             clf_MLP.compile(optimizer= 'adam', loss='binary_crossentropy')
             return clf_MLP

         # Initialiser les trois modèles avec les paramètres par défaut
         clf_A = LogisticRegression(random_state = 0)
         clf_B = DecisionTreeClassifier(random_state = 0)
         clf_C = GradientBoostingClassifier(random_state=0)
         clf_MLP = KerasClassifier(build_fn=keras_model, verbose=0)

         # Calculez le nombre d'échantillons pour 1%, 10% et 100% des données d'entraînement
         samples_100 = len(data['y_train'])
         samples_10 = int(round(0.1*samples_100))
         samples_1 = int(round(0.1*samples_10))

         # Collecter les résultats sur les apprenants
         results = {}
         for clf in [clf_A, clf_B, clf_C, clf_MLP]:
             clf_name = clf.__class__.__name__
             results[clf_name] = {}
             for i, samples in enumerate([samples_1, samples_10, samples_100]):
                 results[clf_name][i] = \
                     train_predict(clf, samples,
                                   data_agg['X_train'].values, data['y_train']['in_hospital_death'],
```

```

data_agg['X_test'].values, data['y_test']['in_hospital_death'], s

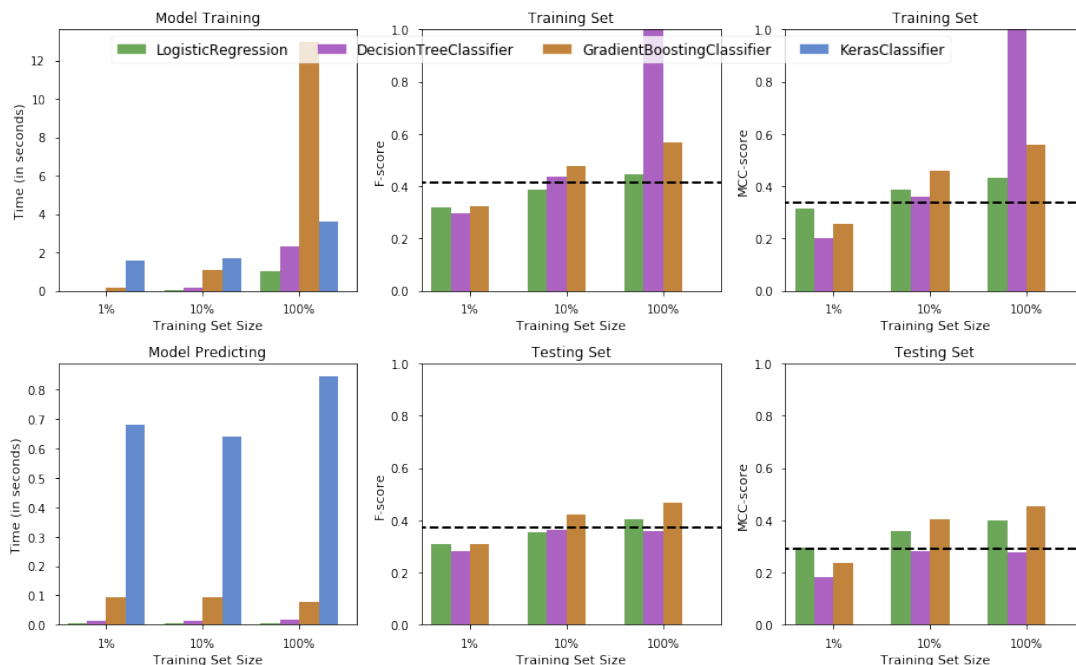
# visualiser les résultats
vs.evaluate(results, score_benchmark)

LogisticRegression formé sur 192 échantillons.
LogisticRegression formé sur 1922 échantillons.
LogisticRegression formé sur 19220 échantillons.
DecisionTreeClassifier formé sur 192 échantillons.
DecisionTreeClassifier formé sur 1922 échantillons.
DecisionTreeClassifier formé sur 19220 échantillons.
GradientBoostingClassifier formé sur 192 échantillons.
GradientBoostingClassifier formé sur 1922 échantillons.
GradientBoostingClassifier formé sur 19220 échantillons.

/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning:
  'precision', 'predicted', average, warn_for)
/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:538: RuntimeWarning: in
  mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)

KerasClassifier formé sur 192 échantillons.
KerasClassifier formé sur 1922 échantillons.
KerasClassifier formé sur 19220 échantillons.

```



Réglage des hyperparamètres Nous avons mis en place une fonction pour utiliser sklearn Grid-Search pour régler les hyperparamètres.

```

In [35]: # Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer

def gridsearch(clf, scorer, X_train, y_train, X_test, y_test, parameters):
    """ apply grid search on clf """
    # Effectuez une recherche de grille sur le classificateur en utilisant 'scorer' comme m

    # Ajoute l'objet de recherche de grille aux données d'apprentissage et trouve les param

    # Obtenez le meilleur estimateur, paramètres et cv_results    best_clf = grid_fit.best_
    best_score = grid_fit.best_score_
    best_params = grid_fit.best_params_

    cv_results = grid_fit.cv_results_
    means = cv_results['mean_test_score']
    stds = cv_results['std_test_score']
    params = cv_results['params']

    print("Best parameters: {}".format(best_params))
    print("Best score: {}".format(best_score))
    for mean, stdev, param in zip(means, stds, params):
        print("mean: {:.3f}, std: {:.3f} for {}".format(mean, stdev, param))

    return best_score, best_params, cv_results

```

Réglage sur le classificateur boost boost.

```

In [34]: # initialiser le classificateur
clf = GradientBoostingClassifier(random_state = 0)

# Créer un objet de scoring fbeta_score en utilisant make_scorer ()
scorer = make_scorer(f1_score)

# Créez la liste des paramètres que vous souhaitez régler, en utilisant un dictionnaire
# nous accordons d'abord le taux d'apprentissage du classificateur
learning_rate = np.arange(0.1, 0.31, 0.1)
n_estimators = np.arange(60,101,20)
parameters = dict(learning_rate=learning_rate, n_estimators=n_estimators)

best_clf, best_predictions, cv_results = gridsearch(clf, scorer,
                                                    data_agg['X_train'], data['y_train']
                                                    data_agg['X_test'], data['y_test'])
                                                    parameters)

```

-----  
NameError

Traceback (most recent call last)

```

<ipython-input-34-ee4c66f99fd5> in <module>()
    14                                     data_agg['X_train'], data['y_train']
    15                                     data_agg['X_test'], data['y_test']
---> 16                                     parameters)

```

```

<ipython-input-31-d9f9738e88f0> in gridsearch(clf, scorer, X_train, y_train, X_test, y_test)
    10
    11 # Obtenez le meilleur estimateur, paramètres et cv_results      best_clf = grid_fit.best_estimator_
---> 12     best_score = grid_fit.best_score_
    13     best_params = grid_fit.best_params_
    14

```

NameError: name 'grid\_fit' is not defined

```

In [25]: # learning rate = 0.2 in this case
         clf = GradientBoostingClassifier(random_state = 0)

         learning_rate = [0.2]
         n_estimators = [100]
         max_depth = np.arange(3, 12, 2)
         min_samples_split = np.arange(100, 1001, 200)

         parameters = dict(learning_rate=learning_rate, n_estimators=n_estimators,
                             max_depth=max_depth, min_samples_split=min_samples_split)

         best_clf, best_predictions, cv_results = gridsearch(clf, scorer,
                                                             data_agg['X_train'], data['y_train']['in_hospital'],
                                                             data_agg['X_test'], data['y_test']['in_hospital'],
                                                             parameters)

```

NameError

Traceback (most recent call last)

```

<ipython-input-25-58020eaf9456> in <module>()
    13                                     data_agg['X_train'], data['y_train']['in_hospital'],
    14                                     data_agg['X_test'], data['y_test']['in_hospital'],
---> 15                                     parameters)

```

```

<ipython-input-23-d9f9738e88f0> in gridsearch(clf, scorer, X_train, y_train, X_test, y_test)
    10
    11 # Obtenez le meilleur estimateur, paramètres et cv_results      best_clf = grid_fit.best_estimator_

```



```

---> 12     best_score = grid_fit.best_score_
      13     best_params = grid_fit.best_params_
      14

```

NameError: name 'grid\_fit' is not defined

In [29]: *# learning rate = 0.25 in this case*

```

      clf = GradientBoostingClassifier(random_state = 0)

```

```

      learning_rate = [0.2]

```

```

      n_estimators = [100]

```

```

      max_depth = [5]

```

```

      min_samples_split = [100]

```

```

      min_samples_leaf = np.arange(1, 81, 20)

```

```

      parameters = dict(learning_rate=learning_rate, n_estimators=n_estimators,
                        max_depth=max_depth, min_samples_split=min_samples_split,
                        min_samples_leaf=min_samples_leaf)

```

```

      best_clf, best_predictions, cv_results = gridsearch(clf, scorer,
                                                         data_agg['X_train'], data['y_train']['in_hospital_death'],
                                                         data_agg['X_test'], data['y_test']['in_hospital_death'],
                                                         parameters)

```

Best parameters: {'learning\_rate': 0.2, 'max\_depth': 5, 'min\_samples\_leaf': 1, 'min\_samples\_split': 100}

Best score: 0.506621848726305

mean: 0.506622, std: 0.010116 for {'learning\_rate': 0.2, 'max\_depth': 5, 'min\_samples\_leaf': 1, 'min\_samples\_split': 100}

mean: 0.501289, std: 0.021659 for {'learning\_rate': 0.2, 'max\_depth': 5, 'min\_samples\_leaf': 21, 'min\_samples\_split': 100}

mean: 0.493081, std: 0.010773 for {'learning\_rate': 0.2, 'max\_depth': 5, 'min\_samples\_leaf': 41, 'min\_samples\_split': 100}

mean: 0.493425, std: 0.021709 for {'learning\_rate': 0.2, 'max\_depth': 5, 'min\_samples\_leaf': 61, 'min\_samples\_split': 100}

In [30]: tuned\_GB\_clf = GradientBoostingClassifier(random\_state = 0, learning\_rate=0.2, n\_estimators=100, max\_depth=5, min\_samples\_split=100, min\_samples\_leaf=1)

```

      tuned_GB_clf.fit(data_agg['X_train'].values, data['y_train']['in_hospital_death'])

```

```

      predictions_test = tuned_GB_clf.predict(data_agg['X_test'].values)

```

```

      tuned_GB_clf_results = {}

```

```

      for score, func in score_metrics.items():

```

```

          # Compute score on the train subset and test set

```

```

              tuned_GB_clf_results[score+"_test"] = func(data['y_test']['in_hospital_death'], predictions_test)

```

```

      print('tuned gradient boost:', tuned_GB_clf_results)

```

tuned gradient boost: {'f1\_test': 0.48437500000000006, 'mcc\_test': 0.4485994263811551}

```

In [31]: # fix random seed for reproducibility
seed = 0
np.random.seed(seed)

# create keras wrapper
def keras_model():
    clf_MLP = Sequential()
    clf_MLP.add(Dense(128, activation='relu', input_shape= data_agg['X_train'].shape[1:]))
    clf_MLP.add(Dropout(0.1))
    clf_MLP.add(Dense(32, activation='relu'))
    clf_MLP.add(Dropout(0.1))
    clf_MLP.add(Dense(8, activation='relu'))
    clf_MLP.add(Dropout(0.1))
    clf_MLP.add(Dense(1, activation='sigmoid'))
    clf_MLP.compile(optimizer= 'adam', loss='binary_crossentropy')
    return clf_MLP

clf = KerasClassifier(build_fn=keras_model, verbose=0)

# Make an f1_score scoring object using make_scorer()
scorer = make_scorer(f1_score)

# define the grid search parameters
batch_size = [32, 256, 512]
epochs = [10, 50, 100]
parameters = dict(batch_size=batch_size, epochs=epochs)

best_clf, best_predictions, cv_results = gridsearch(clf, scorer,
                                                    data_agg['X_train'].values, data['y_train']['in_h'],
                                                    data_agg['X_test'].values, data['y_test']['in_h'],
                                                    parameters)

Best parameters: {'batch_size': 32, 'epochs': 10}
Best score: 0.49446741356509083
mean: 0.494467, std: 0.034073 for {'batch_size': 32, 'epochs': 10}
mean: 0.471002, std: 0.015658 for {'batch_size': 32, 'epochs': 50}
mean: 0.456275, std: 0.018518 for {'batch_size': 32, 'epochs': 100}
mean: 0.481420, std: 0.019687 for {'batch_size': 256, 'epochs': 10}
mean: 0.491392, std: 0.032887 for {'batch_size': 256, 'epochs': 50}
mean: 0.481981, std: 0.018899 for {'batch_size': 256, 'epochs': 100}
mean: 0.448962, std: 0.013769 for {'batch_size': 512, 'epochs': 10}
mean: 0.492628, std: 0.022415 for {'batch_size': 512, 'epochs': 50}
mean: 0.465478, std: 0.015190 for {'batch_size': 512, 'epochs': 100}

In [32]: # fix random seed for reproducibility
seed = 0
np.random.seed(seed)

```

```

# create model
tuned_MLP_clf = Sequential()
tuned_MLP_clf.add(Dense(128, activation='relu', input_shape= data_agg['X_train'].shape[1:]))
tuned_MLP_clf.add(Dropout(0.1))
tuned_MLP_clf.add(Dense(32, activation='relu'))
tuned_MLP_clf.add(Dropout(0.1))
tuned_MLP_clf.add(Dense(8, activation='relu'))
tuned_MLP_clf.add(Dropout(0.1))
tuned_MLP_clf.add(Dense(1, activation='sigmoid'))
tuned_MLP_clf.compile(optimizer='adam', loss='binary_crossentropy')

# train the model
if not os.path.exists('data'):
    os.makedirs('data')
tuned_MLP_clf.fit(data_agg['X_train'].values, data['y_train']['in_hospital_death'],
                  batch_size=512, epochs=50, verbose=1, shuffle=True)

# predict the labels
predictions_test = tuned_MLP_clf.predict(data_agg['X_test'].values) > 0.5

# record the results
tuned_MLP_clf_results = {}
for score, func in score_metrics.items():
    tuned_MLP_clf_results[score+"_test"] = func(data['y_test']['in_hospital_death'], pr

print('tuned MLP:', tuned_MLP_clf_results)

```

```

Epoch 1/50
19220/19220 [=====] - 3s 163us/step - loss: 0.3992
Epoch 2/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2966
Epoch 3/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2734
Epoch 4/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2662
Epoch 5/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2630
Epoch 6/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2588
Epoch 7/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2564
Epoch 8/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2555
Epoch 9/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2540
Epoch 10/50
19220/19220 [=====] - 0s 13us/step - loss: 0.2504

```

Epoch 11/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2501  
Epoch 12/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2514  
Epoch 13/50  
19220/19220 [=====] - 0s 15us/step - loss: 0.2502  
Epoch 14/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2473  
Epoch 15/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2460  
Epoch 16/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2473  
Epoch 17/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2437  
Epoch 18/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2411  
Epoch 19/50  
19220/19220 [=====] - 0s 13us/step - loss: 0.2372  
Epoch 20/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2373  
Epoch 21/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2355  
Epoch 22/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2331  
Epoch 23/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2320  
Epoch 24/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2313  
Epoch 25/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2318  
Epoch 26/50  
19220/19220 [=====] - 0s 16us/step - loss: 0.2281  
Epoch 27/50  
19220/19220 [=====] - 0s 15us/step - loss: 0.2303  
Epoch 28/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2252  
Epoch 29/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2271  
Epoch 30/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2245  
Epoch 31/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2239  
Epoch 32/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2214  
Epoch 33/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2193  
Epoch 34/50  
19220/19220 [=====] - 0s 14us/step - loss: 0.2209

```

Epoch 35/50
19220/19220 [=====] - 0s 15us/step - loss: 0.2195
Epoch 36/50
19220/19220 [=====] - 0s 15us/step - loss: 0.2164
Epoch 37/50
19220/19220 [=====] - 0s 17us/step - loss: 0.2161
Epoch 38/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2166
Epoch 39/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2147
Epoch 40/50
19220/19220 [=====] - 0s 15us/step - loss: 0.2135
Epoch 41/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2165
Epoch 42/50
19220/19220 [=====] - 0s 13us/step - loss: 0.2112
Epoch 43/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2093
Epoch 44/50
19220/19220 [=====] - 0s 13us/step - loss: 0.2124
Epoch 45/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2084
Epoch 46/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2040
Epoch 47/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2060
Epoch 48/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2069
Epoch 49/50
19220/19220 [=====] - 0s 14us/step - loss: 0.2065
Epoch 50/50
19220/19220 [=====] - 0s 15us/step - loss: 0.2023
tuned MLP: {'f1_test': 0.4902386117136659, 'mcc_test': 0.4486404596797615}

```

Importance des fonctionnalités Nous voulons savoir comment la caractéristique liée à la mort à l'hôpital.

```

In [33]: # Train the supervised model on the training set using .fit(X_train, y_train)
         model = GradientBoostingClassifier(random_state = 0, learning_rate=0.2, n_estimators=100,
         max_depth=5, min_samples_split=100, min_samples_leaf=5)

         model.fit(data_agg['X_train'], data['y_train']['in_hospital_death'])

         # Extract the feature importances using .feature_importances_
         importances = model.feature_importances_

         # Plot
         vs.feature_plot(importances, data_agg['X_train'], data['y_train']['in_hospital_death'])

```

