Année Académique : 2023-2024

MASTER II

MODELISATION, CALCUL ET AIDE A LA DECISION

Statistiques des données de grandes dimensions

Projet :

PREDICTION DE LA SURVIE DES PATIENTS

19/12/2023

Enseignant :	*Etudiants* :
Anne Gegout-Petit	Vanga Gustave Hermann MOULO
	Pascaline Kouda

Table des matières

1 Introduction

Le contexte de l'état de santé général d'un patient s'est avéré particulièrement important pendant la pandémie de COVID-19, alors que les professionnels de santé du monde entier sont aux prises avec des hôpitaux surchargés de patients dans un état critique. Les unités de soins intensifs (USI) manquent souvent d'antécédents médicaux vérifiés pour les patients entrants. Un patient en détresse ou un patient amené confus ou inconscient peut ne pas être en mesure de fournir des informations sur des maladies chroniques telles que des maladies cardiaques, des blessures ou le diabète. Le transfert des dossiers médicaux peut prendre des jours, en particulier pour un patient provenant d'un autre prestataire ou système médical.
De plus, La connaissance des maladies chroniques peut éclairer les décisions cliniques concernant les soins des patients et, enfin de compte, améliorer les résultats en matière de survie des patients.

L'identification de la probabilité de mortalité des patients peut aider les médecins à reconnaître les facteurs de risque particuliers de décès des patients et à traiter les cas à risque plus élevé avec des soins plus intensifs. Les taux de mortalité des patients sont couramment utilisés pour mesurer les performances des hôpitaux et l'objectif global de tout centre médical est de réduire le taux de mortalité des patients.
Être capable de prédire avec précision la survie d'un patient en fonction de ses signes vitaux et des facteurs saisis dans ses dossiers est l'une des premières mesures à prendre pour améliorer les soins aux patients et augmenter leur survie.

Objectif du projet :
Développer et valider un modèle de prédiction de la survie chez les patients admis.

2 Analyse exploratoire des données

Notre étude est basée sur des données des patients dans les unités de soins coronariens réalisée en 2021 et compilé par Mitisha Agarwal présent sur kaggle.

Dans cet ensemble de données, divers facteurs sont indiqués qui sont impliqués lorsqu'un patient est hospitalisé. Sur la base de ces facteurs, nous prédisons si le patient survivra ou non.

L'analyse descriptive nous a donné les résultats :
- 59 variables quantitatives
- 25 variables qualitatives dont la variable à prédire qui est **hospital_death**
- Une variable vide qui 'Unnamed : 83'
- 91713 observations

Description de la variable cible Notre variable cible **hospital_death** est qualitative binaire (0 : Survie du patient, 1 : mort du patient).
Nous avons 83798 patients qui sont en vie soit 91,73% de l'ensemble des individus total contre 7915 des patients qui sont morts soit 8,63%
Il s'agit d'un cas de Imbalanced data c'est à dire données déséquilibrées. Pour rééquilibrer notre ensemble de données pour pouvons utiliser la méthode de undersampling car nous avons plus de 10000 observations.

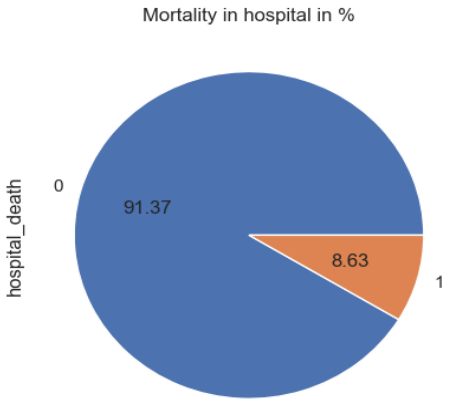

FIGURE 1 – Diagramme de hospital_death %

3 Préparation des données

3.1 Valeurs aberrantes

En réalisant des box-plots pour chaque variable, on remarque toutes les variables quantitatives ont des valeurs aberrantes exceptés les variables représentant les identités.
Le pourcentage des observations contenant les valeurs aberrantes est d'environ de 20%. Comme nous n'avons pas beaucoup d'information sur notre jeu de donnée, nous n'allons pas les traiter.

3.2 Valeurs manquantes

Notre jeu de donnée contient 3,7% des valeurs manquantes totales. Les variables qui contiennent le plus de données manquantes sont les variables d1_potassium_min et d1_potassium_max. Le pourcentage des données qui manquent pour ces variables est 10,85%. Pour les traiter,nous allons les remplacer par la médiane pour chaque variable. Ce choix est fait à cause de la présence de valeurs aberrantes dans le jeu de donnée et cela minimise leur impact lors de l'imputation.

3.3 Transformation des variables

Afin de préparer notre jeu de donnée pour la modélisation, nous avons procéder à remplacer les modalités des variables qualitatives de type object par des entiers car Scikit-learn, la bibliothèque que l'on va utiliser pour modéliser rejette ce type de variable.
Aussi nous allons standardiser les données pour assurer que toutes les variables contribuent de manière égale à l'analyse et à la modélisation.

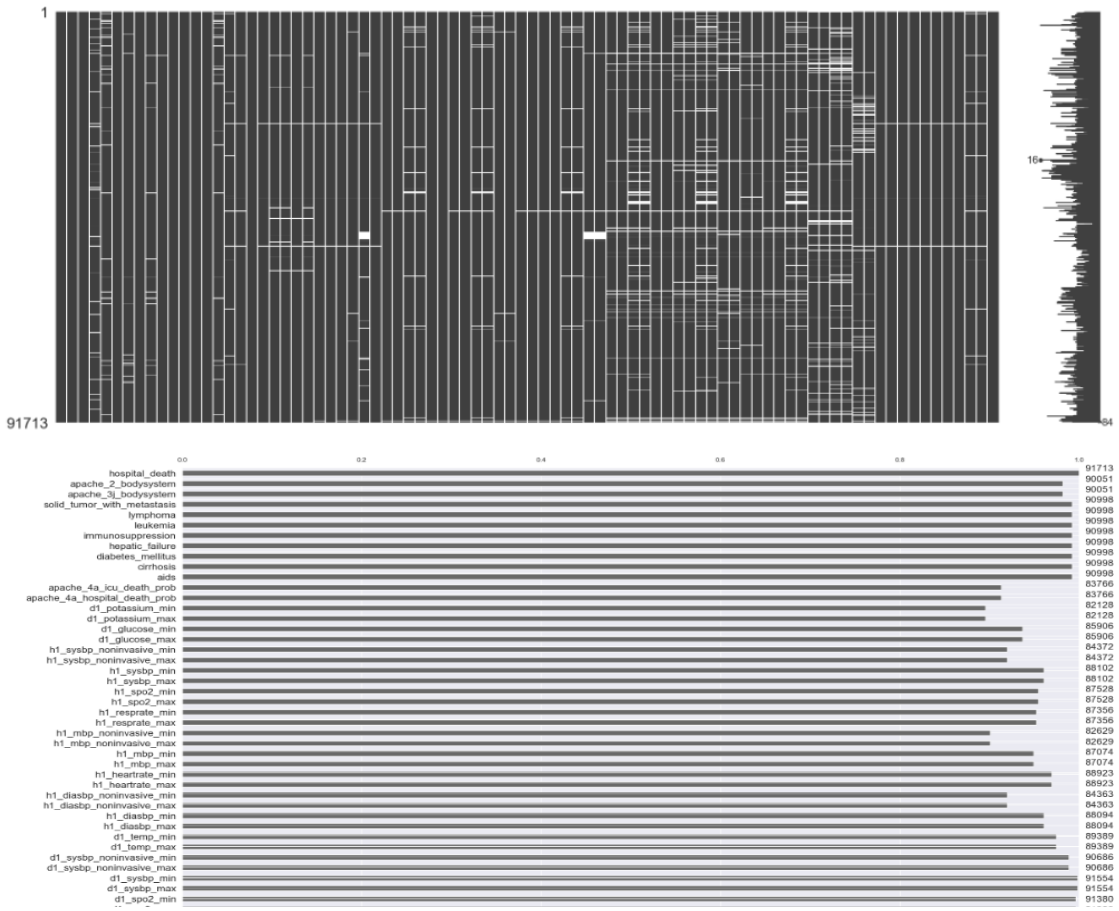

FIGURE 2 – visualisation des données manquantes

4 Sélection des variables

Nous avons supprimer les variables désignant les identités telles que encounter_id,patient_id,hospital_id, icu_id car celles-ci sont sans effet significatif sur l'analyse.
Puis, nous avons fait les tests d'indépendance entre chaque variable explicative et la variable cible et stocker les p-valeurs pour l'ajustement pour une meilleure sélection des caractéristiques. Nous avons utilisé ici l'ajustement de Sidak et de Bonferroni qui nous ont permis de supprimer 8 variables à savoir :d1_diasbp_max, d1_diasbp_noninvasive_max, d1_temp_max, h1_spo2_max, d1_potassium_min, ethnicity, gender, aids. Notons que les deux méthodes nous ont donné les memes variables non pertinentes.
Il nous reste au total 71 variables explicatives pour la modélisation.

5 Modélisation

Dans cette phase, nous nous appuierons sur les insights tirés de l'exploration des données pour concevoir des modèles capables de résoudre notre problème.
Étant donné que nous sommes en face d'un problème de classification, nous utilisons donc les algorithmes d'apprentissage statistiques qui y sont adaptés. Nous allons par la suite évaluer la performance des différents modèles en utilisant des métriques appropriées.

Pour cela, nous considérons les modèles suivants :
— Régression logistique
— Forêt aléatoire
— Régression logistique avec pénalisation élasticnet
Pour chaque modèle, nous divisons notre jeu de données en données d'entraînement et données de test qui servira à évaluer le modèle à l'aide de train_test_split de Sckit-learn.

Traitement des données déséquilibrées(imbalanced data)
On remarque qu'il y a un déséquilibre au niveau de la répartition de la variable qui prédit 91% de modalité 0 contre 9% pour la modalité 1.
Pour rééquilibrer notre ensemble de données d'entraînement, nous avons utilisé la méthode "undersampling" qui est une technique courante de gestion des classes déséquilibrées dans un ensemble de données, où une classe est représentée par un nombre beaucoup plus faible d'instances par rapport à l'autre classe. L'objectif de l'undersampling est de réduire la sur-représentation de la classe majoritaire en réduisant son nombre d'instances, équilibrant ainsi les classes.

Après avoir équilibrer l'ensemble d'entrainement, nous avons obtenu 6328 modalités de 0 et 6328 modalités de 1.

Par la suite nous faisons deux fois la modélisation, une sans le rééquilibrage et l'autre avec.
Nous obtenons les résultats suivant pour la régression logistique.

5.1 Régression logistique

La régression logistique est un modèle de régression utilisé pour la classification binaire. Elle modélise la probabilité qu'une observation appartienne à une classe spécifique en utilisant la fonction logistique.

Metriques sans undersampled

	Métrique	Ensemble d'entraînement	Ensemble de test
0	Accuracy	0.924642	0.922041
1	AUC	0.623597	0.614209
2	Recall	0.259798	0.241966

Metriques avec undersampled

	Métrique	Ensemble d'entraînement	Ensemble de test
0	Accuracy	0.787137	0.792291
1	AUC	0.787137	0.784486
2	Recall	0.769595	0.775047

FIGURE 3 – Résultats métriques pour régression logistique

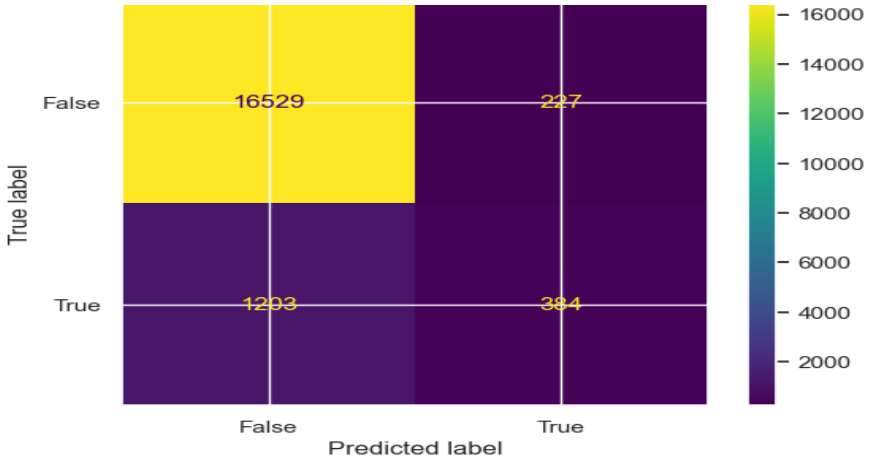

FIGURE 4 – Matrice de confusion

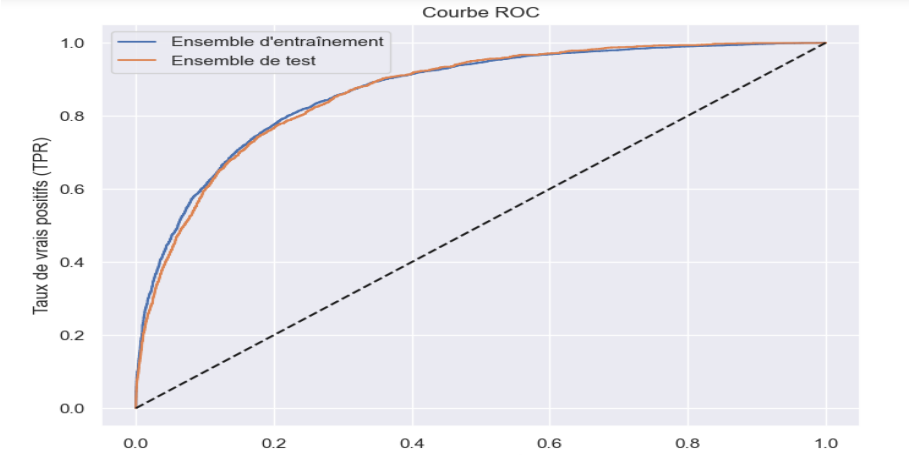

FIGURE 5 – Courbe ROC régression logistique

5.2 Forêt aléatoire (Random forest)

La forêt aléatoire est un modèle d'ensemble qui combine les prédictions de plusieurs arbres de décision. Chaque arbre est construit sur un échantillon aléatoire de données, et les prédictions sont agrégées pour obtenir une prédiction finale. Les forêts aléatoires sont robustes, peu susceptibles de sur-ajuster, et peuvent gérer des ensembles de données complexes avec de nombreuses caractéristiques.

Nous obtenons les résultats suivant pour le Random Forest.

Metriques sans undersampled

	Métrique	Ensemble d'entraînement	Ensemble de test
0	Accuracy	0.999986	0.927111
1	AUC	0.999921	0.630105
2	Recall	0.999842	0.270951

Metriques avec undersampled

	Métrique	Ensemble d'entraînement	Ensemble de test
0	Accuracy	0.999921	0.796489
1	AUC	0.999921	0.801045
2	Recall	0.999842	0.806553

FIGURE 6 – Resultats métriques Random forest

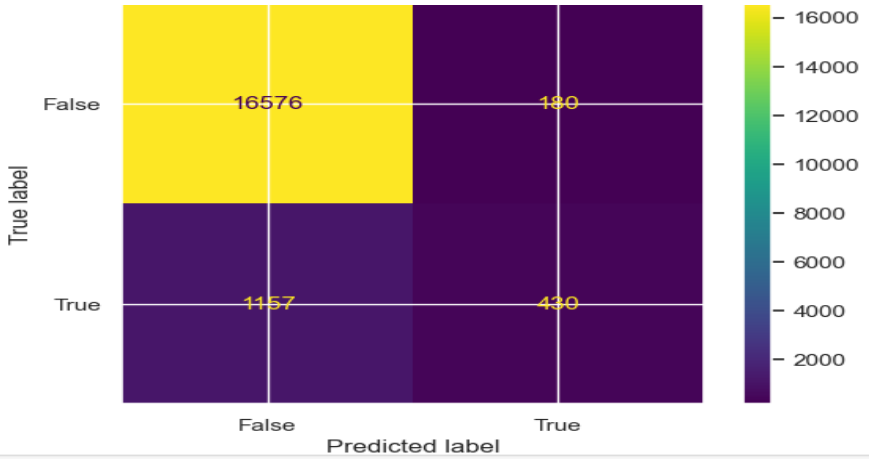

FIGURE 7 – Matrice de confusion

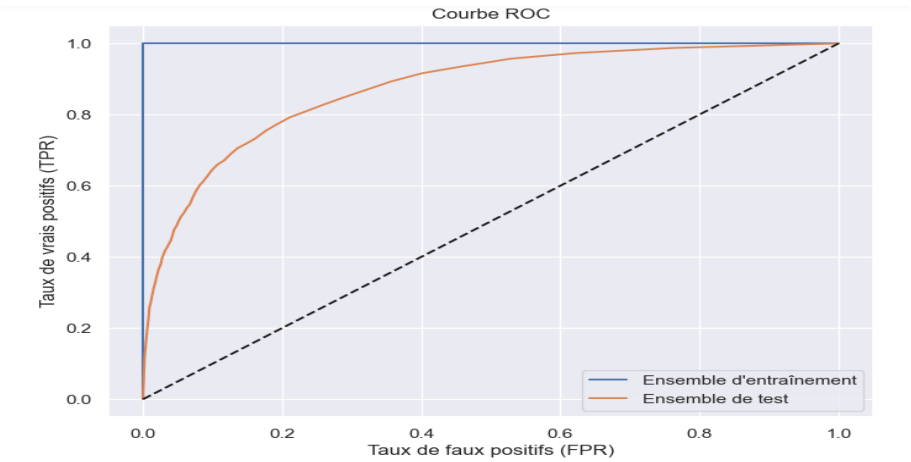

FIGURE 8 – Courbe ROC

5.3 Régression logistique avec pénalisation élasticnet

La régression logistique avec pénalité ElasticNet est une extension de la régression logistique qui incorpore des termes de régularisation L1 (lasso) et L2 (ridge). Elle est utilisée pour traiter des ensembles de données avec de nombreuses caractéristiques en encourageant la sparsité et en contrôlant l'effet des caractéristiques. Cela peut être particulièrement utile pour la sélection automatique des caractéristiques et la gestion des problèmes de multicolinéarité.

Nous obtenons les résultats suivant pour la régression logistique avec pénalisation élasticnet.

Metriques sans undersampled

	Métrique	Ensemble d'entraînement	Ensemble de test
0	Accuracy	0.924547	0.922096
1	AUC	0.622400	0.615095
2	Recall	0.257269	0.243856

Metriques avec undersampled

	Métrique	Ensemble d'entraînement	Ensemble de test
0	Accuracy	0.786188	0.791474
1	AUC	0.786188	0.784894
2	Recall	0.767857	0.776938

FIGURE 9 – Résultats métriques

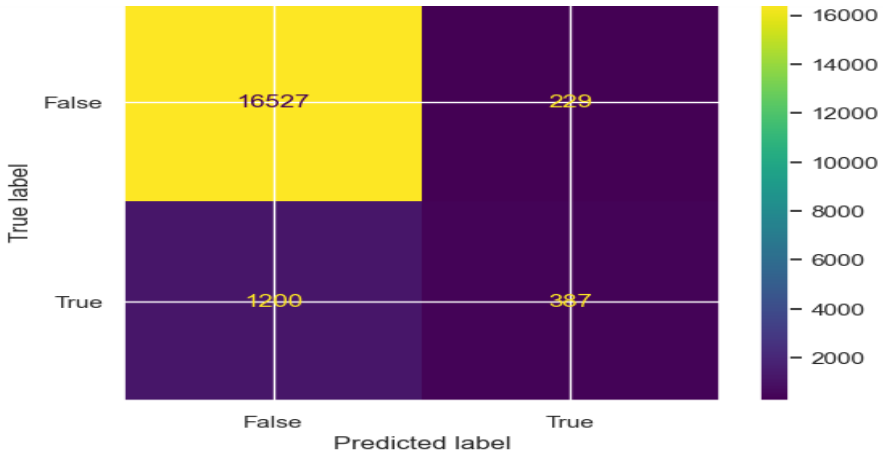

FIGURE 10 – Matrice de confusion

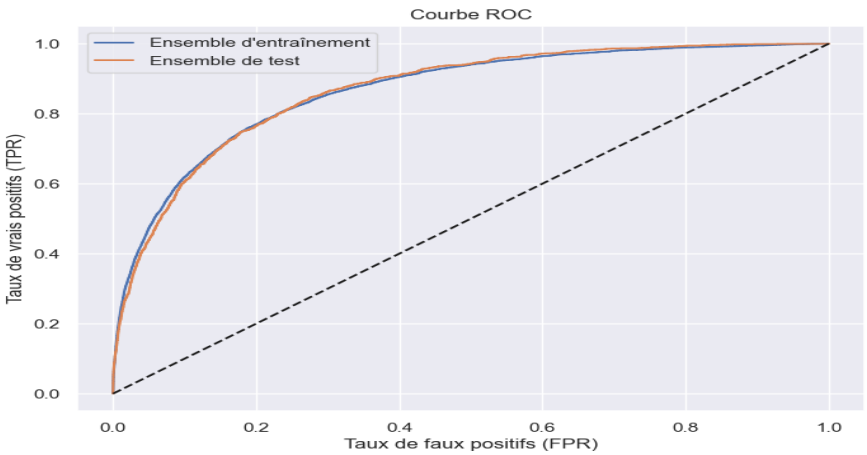

FIGURE 11 – Courbe ROC

6 Évaluation des modèles

— **Accuracy :** La Régression Logistique et la Régression Logistique avec Pénalisation ElasticNet montrent des performances similaires, tandis que Random Forest a une accuracy plus élevée.
— **AUC :** Random Forest montre une amélioration significative de l'AUC par rapport aux modèles de régression logistique, indiquant une meilleure capacité à discriminer entre les classes.
— **Recall :** Random Forest présente un rappel plus élevé dans les deux configurations, ce qui signifie qu'il est meilleur pour identifier les observations réellement positives.

Choix du Meilleur Modèle : En se basant uniquement sur l'Accuracy, Random Forest semble être le meilleur modèle. En considérant l'AUC, Random Forest semble encore être le meilleur modèle. En mettant l'accent sur le Recall, Random Forest est également le meilleur. Ce qui signifie qu'il est plus performant pour identifier les observations positives.

En conclusion, Random Forest semble être le meilleur modèle parmi les trois en termes d'AUC et de Recall, ce qui suggère une meilleure capacité à discriminer entre les classes et à identifier les observations positives.

Le Random Forest nous permet des mesurer l'importance des variables. l'importance des variables dans un modèle de forêt aléatoire fournit des informations essentielles sur les facteurs qui influent le plus sur les prédictions du modèle, ce qui peut être précieux pour l'interprétation et la compréhension du processus de prédiction.

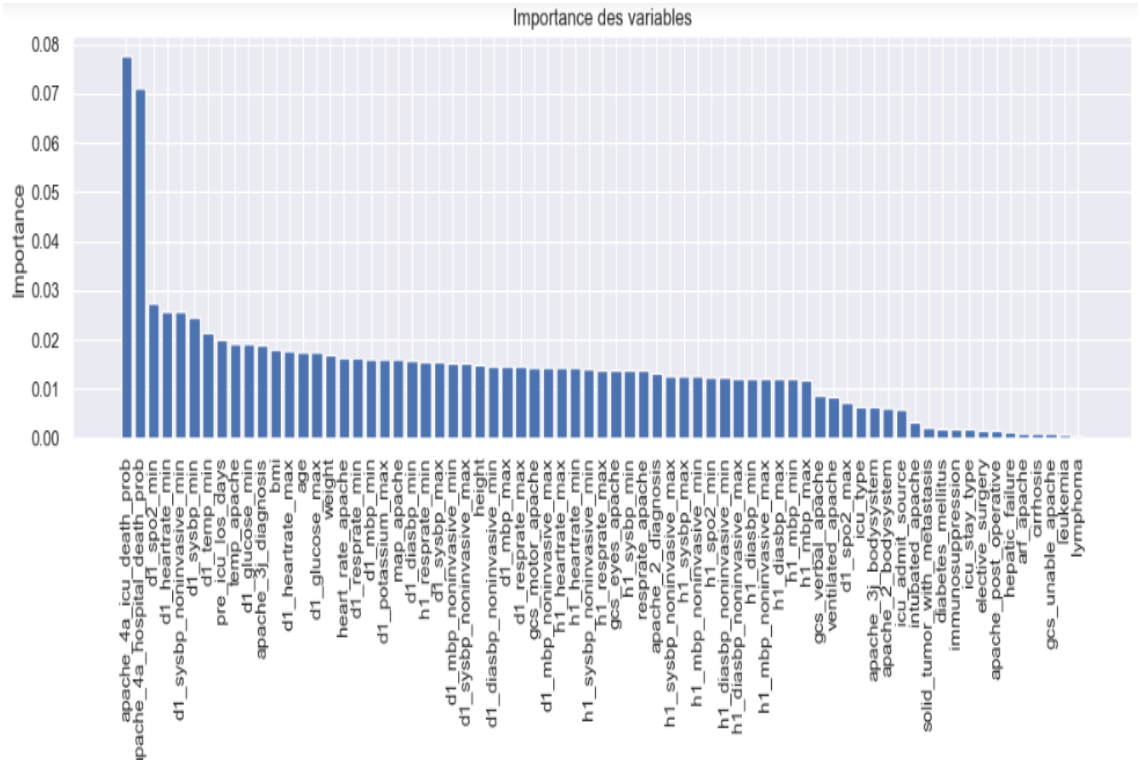

FIGURE 12 – Importance des variables

7 Conclusion

Dans notre travail, nous avons développer et valider un modèle de prédiction de la mortalité hospitalière toutes causes confondues chez les patients admis. Après évaluation de plusieurs algorithmes d'apprentissage statistiques, nous avons retenu Random comme meilleur modèle avec l'undersampling pour prédire la survie des patients, en se basant sur l'AUC qui est de 79.99% et le Recall qui est 80.47%.
Les variables apache_4a_icu_death_prob et apache_4a_hospital_death_prob sont celles qui ont contribué de manière significative à la prédiction de la suivie.

Annexes

```

# Importer les modules nécessaires
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as mnso
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
import statsmodels.api as sm
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from scipy.stats import kruskal
sns.set_theme(style="darkgrid")
from sklearn.feature_selection import SelectKBest,chi2, f_classif
from sklearn.preprocessing import KBinsDiscretizer
from scipy.stats import chi2_contingency
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_
    matrix
from sklearn.preprocessing import StandardScaler
```

```
#Description des noms des variables

dict_description={'encounter_id' : 'Unique identifier associated with a
    patient unit stay',
'patient_id': 'Unique identifier associated with a patient',
'hospital_id': 'Unique identifier associated with a hospital',
'age': 'The age of the patient on unit admission',
'bmi': 'The body mass index of the person on unit admission',
'elective_surgery':'Whether the patient was admitted to the hospital for an
    elective surgical operation',
'ethnicity':'The common national or cultural tradition which the person
    belongs to',
'gender':'Sex of the patient',
'height':'The height of the person on unit admission',
'icu_admit_source':'The location of the patient prior to being admitted to the
     unit',
'icu_id':' unique identifier for the unit to which the patient was admitted',
'icu_stay_type':'string',
'icu_type': 'A classification which indicates the type of care the unit is
    capable of providing',
'pre_icu_los_days': 'The length of stay of the patient between hospital
    admission and unit admission',
'weight' :'The weight (body mass) of the person on unit admission',
'apache_2_diagnosis' :'The APACHE II diagnosis for the ICU admission',
'apache_3j_diagnosis' :'The APACHE III-J sub-diagnosis code which best
    describes the reason for the ICU admission',
'apache_post_operative':'The APACHE operative status; 1 for post-operative, 0
    for non-operative',
```

```
'arf_apache':' Whether the patient had acute renal failure during the first 24 hours of their unit stay, defined as a 24 hour urine output <410ml, creatinine >=133 micromol/L and no chronic dialysis',
'gcs_eyes_apache':'The eye opening component of the Glasgow Coma Scale measured during the first 24 hours which results in the highest APACHE III score',
'gcs_motor_apache': 'The motor component of the Glasgow Coma Scale measured during the first 24 hours which results in the highest APACHE III score',
'gcs_unable_apache': 'Whether the Glasgow Coma Scale was unable to be assessed due to patient sedation',
 'gcs_verbal_apache': 'The verbal component of the Glasgow Coma Scale measured during the first 24 hours which results in the highest APACHE III score',
'heart_rate_apache': 'The heart rate measured during the first 24 hours which results in the highest APACHE III score',
'intubated_apache': 'Whether the patient was intubated at the time of the highest scoring arterial blood gas used in the oxygenation score',
'map_apache': 'The mean arterial pressure measured during the first 24 hours which results in the highest APACHE III score',
'resprate_apache': 'The respiratory rate measured during the first 24 hours which results in the highest APACHE III score',
'temp_apache': 'The temperature measured during the first 24 hours which results in the highest APACHE III score',
'ventilated_apache': 'Whether the patient was invasively ventilated at the time of the highest scoring arterial blood gas using the oxygenation scoring algorithm, including any mode of positive pressure ventilation delivered through a circuit attached to an endo-tracheal tube or tracheostomy',
'd1_diasbp_max':"The patient's highest diastolic blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured",
'd1_diasbp_min':"The patient's lowest diastolic blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured",
'd1_diasbp_noninvasive_max':"The patient's highest diastolic blood pressure during the first 24 hours of their unit stay, non-invasively measured",
'd1_diasbp_noninvasive_min':"The patient's lowest diastolic blood pressure during the first 24 hours of their unit stay, non-invasively measured",
'd1_heartrate_max':"The patient's highest heart rate during the first 24 hours of their unit stay",
'd1_heartrate_min':"The patient's lowest heart rate during the first 24 hours of their unit stay",
'd1_mbp_max':"The patient's highest mean blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured",
'd1_mbp_min':"The patient's lowest mean blood pressure during the first 24 hours of their unit stay, either non-invasively or invasively measured",
'd1_mbp_noninvasive_max':"The patient's highest mean blood pressure during the first 24 hours of their unit stay, non-invasively measured",
'd1_mbp_noninvasive_min':"The patient's lowest mean blood pressure during the first 24 hours of their unit stay, non-invasively measured",
'd1_resprate_max':"The patient's highest respiratory rate during the first 24 hours of their unit stay",
'd1_resprate_min':"The patient's lowest respiratory rate during the first 24 hours of their unit stay",
'd1_spo2_max':"The patient's highest peripheral oxygen saturation during the first 24 hours of their unit stay",
```

```
'd1_spo2_min':"The patient's lowest peripheral oxygen saturation during the first 24 hours of their unit stay",
'd1_sysbp_max':"The patient's highest systolic blood pressure :uring the first 24 hours of their unit stay, either non-invasively or invasively measured",
'd1_sysbp_min':"The patient's lowest systolic blood pressure :uring the first 24 hours of their unit stay, either non-invasively or invasively measured",
'd1_sysbp_noninvasive_max':"The patient': highest systolic blood pressure during the first 24 hours of their unit stay, invasively measured",
'd1_sysbp_noninvasive_min':"The patient': lowest systolic blood pressure during the first 24 hours of their unit stay, invasively measured",
'd1_temp_max':"The patient:s highest core temperature during the first 24 hours of their unit stay, invasively measured",
'd1_temp_min':"The patient's lowest core temperature during the first 24 hours of their unit stay",
'h1_diasbp_max':"The patient's highest diastolic blood :ressure during the first hour of their unit stay, either non-invasively or invasively measured",
'h1_diasbp_min':"The patient's lowest diastolic blood :ressure during the first hour of their unit stay, either non-invasively or invasively measured",
'h1_diasbp_noninvasive_max':"The patient:s highest diastolic blood pressure during the first hour of their unit stay, invasively measured",
'h1_diasbp_noninvasive_min':"The patient:s lowest diastolic blood pressure during the first hour of their unit stay, invasively measured",
'h1_heartrate_max':"The patient's highest heart rate during the first hour of their unit stay",
'h1_heartrate_min':"The patient's lowest heart rate during the first hour of their unit stay",
'h1_mbp_max':"The patient's highest mean blood :ressure during the first hour of their unit stay, either non-invasively or invasively measured",
'h1_mbp_min':"The patient's lowest mean blood :ressure during the first hour of their unit stay, either non-invasively or invasively measured",
'h1_mbp_noninvasive_max':"The patient's :ighest mean blood pressure during the first hour of their unit stay, non-invasively measured",
'h1_mbp_noninvasive_min':"The patient's :owest mean blood pressure during the first hour of their unit stay, non-invasively measured",
'h1_resprate_max':"The patient's highest respiratory rate during the first hour of their unit stay",
'h1_resprate_min':"The patient's lowest respiratory rate during the first hour of their unit stay",
'h1_spo2_max':"The patient's highest peripheral oxygen saturation during the first hour of their unit stay",
'h1_spo2_min':"The patient's lowest peripheral oxygen saturation during the first hour of their unit stay",
'h1_sysbp_max':"The patient's highest systolic blood pressure during the first hour of their unit stay, either non-invasively or invasively measured",
'h1_sysbp_min':"The patient's lowest systolic blood pressure during the first hour of their unit stay, either non-invasively or invasively measured",
'h1_sysbp_noninvasive_max':"The patient's highest systolic blood pressure during the first hour of their unit stay, non-invasively measured",
'h1_sysbp_noninvasive_min':"The patient's lowest systolic blood pressure during the first hour of their unit stay, non-invasively measured",
'd1_glucose_max':"The highest glucose concentration of the patient in their serum or plasma during the first 24 hours of their unit stay",
```

```
'd1_glucose_min':"The lowest glucose concentration of the patient in their serum or plasma during the first 24 hours of their unit stay",
'd1_potassium_max':"The highest potassium concentration for the patient in their serum or plasma during the first 24 hours of their unit stay",
'd1_potassium_min':"The lowest potassium concentration for the patient in their serum or plasma during the first 24 hours of their unit stay",
'apache_4a_hospital_death_prob':"The APACHE IVa probabilistic prediction of in-hospital mortality for the patient which utilizes the APACHE III score and other covariates, including diagnosis.",
'apache_4a_icu_death_prob':"The APACHE IVa probabilistic prediction of in ICU mortality for the patient which utilizes the APACHE III score and other covariates, including diagnosis",
'aids':"Whether the patient has a definitive diagnosis of acquired immune deficiency syndrome (AIDS) (not HIV positive alone)",
'cirrhosis':"Whether the patient has a history of heavy alcohol use with portal hypertension and varices, other causes of cirrhosis with evidence of portal hypertension and varices, or biopsy proven cirrhosis. This comorbidity does not apply to patients with a functioning liver transplant.",
'diabetes_mellitus':"Whether the patient has been diagnosed with diabetes, either juvenile or adult onset, which requires medication.",
'hepatic_failure':"Whether the patient has cirrhosis and additional complications including jaundice and ascites, upper GI bleeding, hepatic encephalopathy, or coma.",
'immunosuppression':"Whether the patient has their immune system suppressed within six months prior to ICU admission for any of the following reasons; radiation therapy, chemotherapy, use of non-cytotoxic immunosuppressive drugs, high dose steroids (at least 0.3 mg/kg/day of methylprednisolone or equivalent for at least 6 months).",
'leukemia':"Whether the patient has been diagnosed with acute or chronic myelogenous leukemia, acute or chronic lymphocytic leukemia, or multiple myeloma.",
'lymphoma':"Whether the patient has been diagnosed with non-Hodgkin lymphoma.",
'solid_tumor_with_metastasis':"Whether the patient has been diagnosed with any solid tumor carcinoma (including malignant melanoma) which has evidence of metastasis.",
'apache_3j_bodysystem':"Admission diagnosis group for APACHE III",
'apache_2_bodysystem':"Admission diagnosis group for APACHE II",
'hospital_death':"Whether the patient died during this hospitalization"}
```

```
    #Chargement des données

data = pd.read_csv('dataset.csv')
data.head()


#Description des données

data.info()
data.describe(include="all")
data.isnull().sum() # compter le nombre de valeurs manquantes
data.isnull().sum().sum()/(data.size)*100# Pourcentage des données manquantes
# Verification des instances dupliquées
data.duplicated().sum()

```

```
#Analyse de la variable cible


# les modalités de la variable
data.hospital_death.unique()
# Nonbre d'individus par modalité
data.hospital_death.value_counts()
# Visualisation de la variable
_=data.hospital_death.value_counts().plot(kind='pie',autopct="%.2f",title ='
    Mortality in hospital in %')


#Visualisation des variables explicatives

data1= data.drop(["hospital_death"],axis=1)
for i in data1.columns[0::]:
    print(i.upper())


    print('')
    print(f'Description: {dict_description[i]}')
    print('')

    if data1[i].value_counts().shape[0]>20   :

        plt.figure(figsize=(9,8))
        sns.boxplot(data1[i])
        plt.show()
        plt.hist(data1[i])
        plt.show()


    else:
        data1[i].value_counts().plot(kind='pie', autopct='%1.1f%%')
        plt.axis('equal')
        plt.show()



    print('_____________________________________________________________________
        ____________')



#Detection des valeurs aberrantes

num_vars = ['age','bmi', 'height', 'pre_icu_los_days', 'weight','apache_2_
    diagnosis', 'apache_3j_diagnosis','heart_rate_apache','map_apache','
    resprate_apache', 'temp_apache', 'd1_diasbp_max', 'd1_diasbp_min', 'd1_
    diasbp_noninvasive_max', 'd1_diasbp_noninvasive_min', 'd1_heartrate_max',
    'd1_heartrate_min', 'd1_mbp_max', 'd1_mbp_min', 'd1_mbp_noninvasive_max',
    'd1_mbp_noninvasive_min', 'd1_resprate_max', 'd1_resprate_min', 'd1_spo2_
    max', 'd1_spo2_min', 'd1_sysbp_max', 'd1_sysbp_min', 'd1_sysbp_noninvasive
    _max', 'd1_sysbp_noninvasive_min', 'd1_temp_max', 'd1_temp_min', 'h1_
    diasbp_max', 'h1_diasbp_min', 'h1_diasbp_noninvasive_max', 'h1_diasbp_
    noninvasive_min', 'h1_heartrate_max', 'h1_heartrate_min', 'h1_mbp_max', '
```

```
    h1_mbp_min', 'h1_mbp_noninvasive_max', 'h1_mbp_noninvasive_min', 'h1_
    resprate_max', 'h1_resprate_min', 'h1_spo2_max', 'h1_spo2_min', 'h1_sysbp_
    max', 'h1_sysbp_min', 'h1_sysbp_noninvasive_max', 'h1_sysbp_noninvasive_
    min', 'd1_glucose_max', 'd1_glucose_min', 'd1_potassium_max', 'd1_
    potassium_min', 'apache_4a_hospital_death_prob', 'apache_4a_icu_death_prob
    ',] # données numerique

def detect_outliers_zscore(data, threshold=3):
    z_scores = np.abs((data - data.mean()) / data.std())
    return z_scores > threshold

# Appliquez la détection des valeurs aberrantes    chaque colonne
outliers_dataset = data[num_vars].apply(detect_outliers_zscore)

# Comptez le nombre de valeurs aberrantes par colonne
nombre_outliers_par_colonne = outliers_dataset.sum()

# Triez les colonnes par le nombre de valeurs aberrantes
colonnes_triees_par_outliers = nombre_outliers_par_colonne.sort_values(
    ascending=False)

# Affichez le résultat
    print(colonnes_triees_par_outliers)
#subdataset des valeurs aberrantes
def subdataset_outliers(data, feature):
    q1=np.percentile(data[feature], 25)
    q3=np.percentile(data[feature], 75)
    iqr=q3-q1
    lower_bound= q1-1.5*iqr
    upper_bound= q1+1.5*iqr

    return data[(data[feature] < lower_bound) | (data[feature] > upper_bound)]

for feature in num_vars:
    subdataset_outlier=subdataset_outliers(data, feature)

subdataset_outlier[num_vars]

#Visualisation des données manquantes

#Visualisation avec matrix
mnso.matrix(data)
plt.show()
plt.savefig(fname="missingmatrix")
#Visualisation avec bar
mnso.bar(data)
plt.show()
```

```

#Traitement des valeurs manquantes
#Tableau de valeurs manquantes par colonnes

cols=["encounter_id","patient_id","hospital_id","icu_id"]
data = data.drop(cols, axis=1)

#Tableau de valeurs manquantes par colonnes
```

```
Missings_values = data.isnull().sum().sort_values(ascending=False)
Missing_percent = (Missings_values/len(data))*100
Missing_table = pd.DataFrame({"Missing values": Missings_values,"Percentage":
    Missing_percent})
Missing_table

# Sélectionner les variables numériques
num_vars = ['age','bmi', 'height', 'pre_icu_los_days', 'weight','apache_2_
    diagnosis', 'apache_3j_diagnosis','heart_rate_apache','map_apache','
    resprate_apache', 'temp_apache', 'd1_diasbp_max', 'd1_diasbp_min', 'd1_
    diasbp_noninvasive_max', 'd1_diasbp_noninvasive_min', 'd1_heartrate_max',
    'd1_heartrate_min', 'd1_mbp_max', 'd1_mbp_min', 'd1_mbp_noninvasive_max',
    'd1_mbp_noninvasive_min', 'd1_resprate_max', 'd1_resprate_min', 'd1_spo2_
    max', 'd1_spo2_min', 'd1_sysbp_max', 'd1_sysbp_min', 'd1_sysbp_noninvasive
    _max', 'd1_sysbp_noninvasive_min', 'd1_temp_max', 'd1_temp_min', 'h1_
    diasbp_max', 'h1_diasbp_min', 'h1_diasbp_noninvasive_max', 'h1_diasbp_
    noninvasive_min', 'h1_heartrate_max', 'h1_heartrate_min', 'h1_mbp_max', '
    h1_mbp_min', 'h1_mbp_noninvasive_max', 'h1_mbp_noninvasive_min', 'h1_
    resprate_max', 'h1_resprate_min', 'h1_spo2_max', 'h1_spo2_min', 'h1_sysbp_
    max', 'h1_sysbp_min', 'h1_sysbp_noninvasive_max', 'h1_sysbp_noninvasive_
    min', 'dhttps://www.overleaf.com/project/657841594ca4efe470a1f6931_glucose
    _max', 'd1_glucose_min', 'd1_potassium_max', 'd1_potassium_min', 'apache_4
    a_hospital_death_prob', 'apache_4a_icu_death_prob',] # données numerique

# Sélectionner les variables catégorielles
cat_vars = ['elective_surgery', 'ethnicity', 'gender', 'icu_admit_source', '
    icu_stay_type', 'icu_type', 'apache_post_operative', 'arf_apache', 'gcs_
    eyes_apache', 'gcs_motor_apache', 'gcs_unable_apache', 'gcs_verbal_apache'
    , 'intubated_apache', 'ventilated_apache', 'aids', 'cirrhosis', 'diabetes_
    mellitus', 'hepatic_failure', 'immunosuppression', 'leukemia', 'lymphoma',
     'solid_tumor_with_metastasis', 'apache_3j_bodysystem', 'apache_2_
    bodysystem', 'hospital_death']


for column in num_vars:
    data[column].fillna(data[column].median(), inplace=True)

for column in cat_vars:
    data[column].fillna(data[column].mode()[0], inplace=True)
data.head()

#visualisation des variables explicatives en fonction des modalités de
    hospital_death


# # Sélectionner les variables catégorielles
cat_vars = ['elective_surgery', 'ethnicity', 'gender', 'icu_admit_source', '
    icu_stay_type', 'icu_type', 'apache_post_operative', 'arf_apache', 'gcs_
    eyes_apache', 'gcs_motor_apache', 'gcs_unable_apache', 'gcs_verbal_apache'
    , 'intubated_apache', 'ventilated_apache', 'aids', 'cirrhosis', 'diabetes_
    mellitus', 'hepatic_failure', 'immunosuppression', 'leukemia', 'lymphoma',
     'solid_tumor_with_metastasis', 'apache_3j_bodysystem', 'apache_2_
    bodysystem', 'hospital_death']

# Générer un bar plot pour chaque variable catégorielle
for var in cat_vars:
```

```
    figsize = (20, 20)
    (data.groupby([var, 'hospital_death'])['hospital_death'].count()/data.
        groupby([var])[var].count()).unstack(level=1).plot(kind='bar', stacked
        =True)
    plt.title(var)
    plt.xlabel('Modalités')
    plt.ylabel('Proportion (%)')
    plt.legend(['Survive', 'Death'])
    plt.show()

Générer un box plot pour chaque variable numérique
num_vars = ['age','bmi', 'height', 'pre_icu_los_days', 'weight','apache_2_
    diagnosis', 'apache_3j_diagnosis','heart_rate_apache','map_apache','
    resprate_apache', 'temp_apache', 'd1_diasbp_max', 'd1_diasbp_min', 'd1_
    diasbp_noninvasive_max', 'd1_diasbp_noninvasive_min', 'd1_heartrate_max',
    'd1_heartrate_min', 'd1_mbp_max', 'd1_mbp_min', 'd1_mbp_noninvasive_max',
    'd1_mbp_noninvasive_min', 'd1_resprate_max', 'd1_resprate_min', 'd1_spo2_
    max', 'd1_spo2_min', 'd1_sysbp_max', 'd1_sysbp_min', 'd1_sysbp_noninvasive
    _max', 'd1_sysbp_noninvasive_min', 'd1_temp_max', 'd1_temp_min', 'h1_
    diasbp_max', 'h1_diasbp_min', 'h1_diasbp_noninvasive_max', 'h1_diasbp_
    noninvasive_min', 'h1_heartrate_max', 'h1_heartrate_min', 'h1_mbp_max', '
    h1_mbp_min', 'h1_mbp_noninvasive_max', 'h1_mbp_noninvasive_min', 'h1_
    resprate_max', 'h1_resprate_min', 'h1_spo2_max', 'h1_spo2_min', 'h1_sysbp_
    max', 'h1_sysbp_min', 'h1_sysbp_noninvasive_max', 'h1_sysbp_noninvasive_
    min', 'd1_glucose_max', 'd1_glucose_min', 'd1_potassium_max', 'd1_
    potassium_min', 'apache_4a_hospital_death_prob', 'apache_4a_icu_death_prob
    ',] # données numerique

for var in num_vars:
    sns.boxplot(x='hospital_death', y=var, data=data)
    plt.title(var)
    plt.xlabel('Reponse binaire')
    plt.ylabel('Valeur')
    plt.show()

#test d'hypothèse et analyse de p-valeurs pour la selection des variables

#Normalité
X = data[num_vars].values
y = data["hospital_death"].values
from scipy.stats import f_oneway,shapiro, levene
# Séparer les données en deux groupes en fonction de la variable cible
group0 = X[y == 0]
group1 = X[y == 1]
# Calculer les p-valeurs pour les tests de normalité (Shapiro-Wilk) et de
    Levene
normality_p_values = np.array([shapiro(X[:, i])[1] for i in range(X.shape[1])
    ])
levene_p_values = np.array([levene(group0[:, i], group1[:, i])[1] for i in
    range(X.shape[1])])

# Seuil de significativité
alpha = 0.05

# Vérifier si les tests sont validés (p-valeur >= alpha)
normality_results = normality_p_values >= alpha
```

```
levene_results = levene_p_values >= alpha

# Créer un DataFrame pour présenter les résultats
results_df = pd.DataFrame({
    'Caractéristiques': num_vars,
    'Normalité (p-valeur)': normality_p_values,
    'Normalité (validé)': normality_results,
    'Levene (p-valeur)': levene_p_values,
    'Levene (validé)': levene_results
})

# Afficher le tableau des résultats
results_df

#test de kruskal wallis pour les variables quantitatives
X = data[num_vars].values
y = data["hospital_death"].values
# Séparer les données en deux groupes en fonction de la variable cible
group0 = X[y == 0]
group1 = X[y == 1]

pvaNu = []
for i in range(X.shape[1]):
    pvaNu.append(kruskal(group0[:, i], group1[:, i]).pvalue)

print(pvaNu)
print(len(pvaNu))
results_df_Num = pd.DataFrame({
    'Variable': num_vars,
    'P-valeur': pvaNu
})

results_df_Num

# Initialiser les listes pour stocker les résultats
var_names = []
p_values = []

#test de Chi-deux pour les variables qualitatives
# Parcourir toutes les variables catégorielles
for var in cat_vars:
    # Calculer le tableau de contingence
    contingency_table = pd.crosstab(data['hospital_death'], data[var])
    # Calculer la statistique de test du Chi-deux et la p-valeur
    chi2, p, dof, expected = chi2_contingency(contingency_table)

    # Ajouter les résultats aux listes correspondantes
    var_names.append(var)
    p_values.append(p)


# Créer un DataFrame avec les résultats
results_df_Cat = pd.DataFrame({
    'Variable': var_names,
    'P-valeur': p_values,
})
```

```

# Afficher le tableau des résultats
results_df_Cat

pvals = pvaNu + p_values
#pvals
nomvars=num_vars + var_names
#nomvars

# Ajustement des p-valeure

import statsmodels.stats.multitest as stm

# sidak
comp = (stm.multipletests(pvals, alpha=0.05, method='s', maxiter=1, is_sorted=
    False, returnsorted=False))[0]
comp
List=[]
for i in range(len(comp)):
    if comp[i]==False:
        List.append(nomvars[i])

print("les variables    supprimer avec l'ajustement de sidak ")
List

# holm
comp1=(stm.multipletests(pvals, alpha=0.05, method='b', maxiter=1, is_sorted=
    False, returnsorted=False))[0]
#comp1
List1=[]
for i in range(len(comp1)):
    if comp1[i]==False:
        List1.append(nomvars[i])
print("les variables    supprimer avec l'ajustement de bonferroni ")
List1

# supprimer les variables non pertinentes
data = data.drop(List, axis=1)
```

```
    # Transformation des variables
data["icu_admit_source"] = data["icu_admit_source"].replace("Accident &
    Emergency", 0).replace("Operating Room / Recovery", 1).replace("Floor", 2)
    .replace("Other Hospital", 3).replace("Other ICU", 4)
data["icu_stay_type"] = data["icu_stay_type"].replace("admit", 0).replace("
    transfer", 1).replace("readmit", 2)
data["icu_type"] = data["icu_type"].replace("Med-Surg ICU", 0).replace("MICU",
     1).replace("Neuro ICU", 2).replace("CCU-CTICU", 3).replace("SICU", 4).
    replace("Cardiac ICU", 5).replace("CSICU", 6).replace("CTICU", 7)
data["apache_3j_bodysystem"] = data["apache_3j_bodysystem"].replace("
    Cardiovascular", 0).replace("Neurological", 1).replace("Sepsis", 2).
    replace("Respiratory", 3).replace("Gastrointestinal", 4).replace("
    Metabolic", 5).replace("Trauma", 6).replace("Genitourinary", 7).replace("
    Musculoskeletal/Skin", 8).replace("Hematological", 9).replace("
    Gynecological", 10)
data["apache_2_bodysystem"] = data["apache_3j_bodysystem"].replace("
    Cardiovascular", 0).replace("Neurologic", 1).replace("Respiratory", 2).
```

```
    replace("Gastrointestinal", 3).replace("Metabolic", 4).replace("Trauma",
    5).replace("Undefined diagnoses", 6).replace("Renal/Genitourinary", 7).
    replace("Haematologic", 8).replace("Undefined Diagnoses", 9)

standardisation des variales quantitatives

#Liste des variables retenues apres l'ajustement des p_valeurs
n_vars = ['age','bmi', 'height', 'pre_icu_los_days', 'weight','apache_2_
    diagnosis', 'apache_3j_diagnosis','heart_rate_apache','map_apache','
    resprate_apache', 'temp_apache', 'd1_diasbp_min', 'd1_diasbp_noninvasive_
    min', 'd1_heartrate_max', 'd1_heartrate_min', 'd1_mbp_max', 'd1_mbp_min',
    'd1_mbp_noninvasive_max', 'd1_mbp_noninvasive_min', 'd1_resprate_max', 'd1
    _resprate_min', 'd1_spo2_max', 'd1_spo2_min', 'd1_sysbp_max', 'd1_sysbp_
    min', 'd1_sysbp_noninvasive_max', 'd1_sysbp_noninvasive_min', 'd1_temp_min
    ', 'h1_diasbp_max', 'h1_diasbp_min', 'h1_diasbp_noninvasive_max', 'h1_
    diasbp_noninvasive_min', 'h1_heartrate_max', 'h1_heartrate_min', 'h1_mbp_
    max', 'h1_mbp_min', 'h1_mbp_noninvasive_max', 'h1_mbp_noninvasive_min', '
    h1_resprate_max', 'h1_resprate_min', 'h1_spo2_min', 'h1_sysbp_max', 'h1_
    sysbp_min', 'h1_sysbp_noninvasive_max', 'h1_sysbp_noninvasive_min', 'd1_
    glucose_max', 'd1_glucose_min', 'd1_potassium_max', 'apache_4a_hospital_
    death_prob', 'apache_4a_icu_death_prob',] # données numerique
c_vars= ['elective_surgery', 'icu_admit_source', 'icu_stay_type', 'icu_type',
    'apache_post_operative', 'arf_apache', 'gcs_eyes_apache', 'gcs_motor_
    apache', 'gcs_unable_apache', 'gcs_verbal_apache', 'intubated_apache', '
    ventilated_apache', 'cirrhosis', 'diabetes_mellitus', 'hepatic_failure', '
    immunosuppression', 'leukemia', 'lymphoma', 'solid_tumor_with_metastasis',
     'apache_3j_bodysystem', 'apache_2_bodysystem', 'hospital_death']

scaler = StandardScaler()
data_quanti = scaler.fit_transform(data[n_vars])
data_quanti.shape

# transfromer en dataframe
data_quanti_final = pd.DataFrame(data_quanti,columns=n_vars)

#Nouveau data
data_new = pd.concat([data_quanti_final,data[c_vars]], axis=1)
data_new

############################### MODELISATION
# Sélectionner les variables explicatives et la variable d'intér t
X = data_new.drop('hospital_death', axis=1)
y = data_new["hospital_death"]
y=y.astype("int")

# Diviser la base de données en train et test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)


# Importer les bibliothèques nécessaires
from imblearn.under_sampling import RandomUnderSampler

# Initialiser l'objet RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy='auto', random_state=42)
```

```

# Appliquer l'undersampling sur les données d'apprentissage
X_train_undersampled, y_train_undersampled = rus.fit_resample(X_train, y_train
    )

# Créer un nouveau DataFrame avec les données undersampled
dataSelect_undersampled = pd.concat([X_train_undersampled, y_train_
    undersampled], axis=1)

# Afficher la nouvelle distribution des données
print(dataSelect_undersampled['hospital_death'].value_counts())

#####Regression logistique

model=LogisticRegression(random_state=42)

# Entra nement du modèle
model.fit(X_train, y_train)

# Prédictions sur l'ensemble de test
y_pred = model.predict(X_test)

# Prédire les classes sur les ensembles d'entra nement et de test
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Calculer les mesures de performance
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_auc = roc_auc_score(y_train, y_train_pred)
test_auc = roc_auc_score(y_test, y_test_pred)
train_recall = recall_score(y_train, y_train_pred)
test_recall = recall_score(y_test, y_test_pred)

# Créer le tableau d'évaluation des performances
performance_table = pd.DataFrame({
    'Métrique': ['Accuracy', 'AUC', 'Recall'],
    'Ensemble d\'entra nement': [train_accuracy, train_auc, train_recall],
    'Ensemble de test': [test_accuracy, test_auc, test_recall]
})

# Afficher le tableau d'évaluation des performances
print('Metriques sans undersampled')
print(performance_table)


conM=confusion_matrix( y_test,y_test_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = conM, display_labels =
    [False, True])
cm_display.plot()
plt.show()
\textbf{}


##### Cas des données equilibrées
model=LogisticRegression(random_state=42)
```

```
94
95  # Entra nement du modèle
96  model.fit(X_train_undersampled, y_train_undersampled)
97
98  # Prédictions sur l'ensemble de test
99  y_pred = model.predict(X_test)
100
101 # Prédire les classes sur les ensembles d'entra nement et de test
102 y_train_pred = model.predict(X_train_undersampled)
103 y_test_pred = model.predict(X_test)
104
105 # Calculer les mesures de performance
106 train_accuracy = accuracy_score(y_train_undersampled, y_train_pred)
107 test_accuracy = accuracy_score(y_test, y_test_pred)
108 train_auc = roc_auc_score(y_train_undersampled, y_train_pred)
109 test_auc = roc_auc_score(y_test, y_test_pred)
110 train_recall = recall_score(y_train_undersampled, y_train_pred)
111 test_recall = recall_score(y_test, y_test_pred)
112
113 # Créer le tableau d'évaluation des performances
114 performance_table_Under = pd.DataFrame({
115     'Métrique': ['Accuracy', 'AUC', 'Recall'],
116     'Ensemble d\'entra nement': [train_accuracy, train_auc, train_recall],
117     'Ensemble de test': [test_accuracy, test_auc, test_recall]
118 })
119 # Obtenir les scores de probabilité pour les classes positives
120 y_train_prob = model.predict_proba(X_train_undersampled)[:, 1]
121 y_test_prob = model.predict_proba(X_test)[:, 1]
122
123 # Calculer les taux de faux positifs (FPR) et les taux de vrais positifs (TPR)
124 train_fpr, train_tpr, _ = roc_curve(y_train_undersampled, y_train_prob)
125 test_fpr, test_tpr, _ = roc_curve(y_test, y_test_prob)
126
127 # Tracer les courbes ROC
128 plt.figure(figsize=(8, 6))
129 plt.plot(train_fpr, train_tpr, label='Ensemble d\'entra nement')
130 plt.plot(test_fpr, test_tpr, label='Ensemble de test')
131 plt.plot([0, 1], [0, 1], 'k--')
132 plt.xlabel('Taux de faux positifs (FPR)')
133 plt.ylabel('Taux de vrais positifs (TPR)')
134 plt.title('Courbe ROC')
135 plt.legend()
136 plt.show()
```