Process ML - CODES

1. **Import dataset et librairies**

```
import pandas as pd
import joblib

# Importation d'Imbalanced-learn
from imblearn import ...

!pip install catboost
!pip install lightgbm
!pip install xgboost

from pandas import read_csv # For dataframes
from pandas import DataFrame # For dataframes
from numpy import ravel # For matrices
import matplotlib.pyplot as plt # For plotting data
import seaborn as sns # For plotting data
from sklearn.model_selection import train_test_split # For tr
#from sklearn.neighbors import KNeighborsClassifier # The k-n
from sklearn.feature_selection import VarianceThreshold # Fea
from sklearn.pipeline import Pipeline # For setting up pipeli
# Various pre-processing steps
```

```
from sklearn.preprocessing import Normalizer, RobustScaler, S
from sklearn.preprocessing import PowerTransformer, MaxAbsSca
from sklearn.model_selection import GridSearchCV # For optimi
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_v
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
import warnings
from sklearn.metrics import accuracy_score, precision_score,
from imblearn.over_sampling import RandomOverSampler
warnings.filterwarnings('ignore')
```

- Pandas
- Joblib
- IMB-learn (Imbalanced-learn) : bibliothèque Python conçue pour traiter les problèmes de déséquilibre de classe dans les ensembles de données.
- Warnings (pour empêcher l'affichage d'un message d'erreur)

2 Encoding

One-hot encoding : exemple de code (https://medium.com/@sii-lille/data-science-one-hot-encoding-c59e82b3f0e7)

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
countries = ['China', 'USA', 'France', 'Italie', 'France']
cities = ['Beijing', 'New York', 'Paris', 'Rome', 'Marseill
# Imaginons un DataFrame avec les colonnes 'countries' et '
df = pd.DataFrame({'countries': countries, 'cities': cities
# Créons l'objet OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)
# Utilisons la méthode 'fit_transform' pour effectuer le or
```

```
one_hot_encod = encoder.fit_transform(df[['countries']])
# Concaténons la colonne 'cities' avec les données 'countri
df_new = pd.concat([df['cities'], pd.DataFrame(one_hot_encc
# Affichons le résultat
df_new
```

Ordinal encoding :

```
import numpy as np
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
sizes = ["small", "medium", "large"]
# reshape to 2D array
sizes = np.array(sizes).reshape(-1,1)
encoded = encoder.fit_transform(sizes)
print(encoded)
```

2. **Random over-sampler / random under-sampler (Greg)**

Si déséquilibre de classe dans un dataset, il est **impératif** de le faire car le modèle qu'on va entraîner aura tendance à être biaisé en faveur de la classe majoritaire.

Random-over sampler : méthode de sur-échantillonnage = augmenter le nombre d'instances de la classe minoritaire en ajoutant des copies aléatoires d'observations de cette classe.

Random-under sampler : méthode de sous-échantillonnage = réduire le nombre d'instances de la classe majoritaire en éliminant aléatoirement des observations de cette classe.

🚨 Le suréchantillonnage aléatoire peut conduire au sur-ajustement (overfitting) en raison de la duplication d'observations existantes, tandis que le sous-échantillonnage aléatoire peut conduire à la perte d'informations importantes présentes dans la classe majoritaire. Tester les deux méthodes pour voir celle qui donne le meilleur score.

https://lerekoqholosha9.medium.com/random-oversampling-and-undersampling-for-imbalanced-classification-a4aad406fd72

```
# Définition des x et du y
X=df_liver_disease.drop('Target',axis=1)
y=df_liver_disease['Target']
```

```
# Affichage actuel des classes (malades / sains)
y.value_counts().plot.pie(autopct='%.2f')
```

```
# Rééquilibrage et visualisation des classes après applicatio
from imblearn.under_sampling import RandomUnderSampler
# rus = RandomUnderSampler(sampling_strategy="not minority") 
rus = RandomUnderSampler(sampling_strategy=1) # Numerical val
X_res, y_res = rus.fit_resample(X, y)
ax = y_res.value_counts().plot.pie(autopct='%.2f')
_ = ax.set_title("Under-sampling")

# Rééquilibrage et visualisation des classes après applicatio
from imblearn.over_sampling import RandomOverSampler
#ros = RandomOverSampler(sampling_strategy=1) # Float
ros = RandomOverSampler(sampling_strategy="not majority") # S
X_res, y_res = ros.fit_resample(X, y)
ax = y_res.value_counts().plot.pie(autopct='%.2f')
_ = ax.set_title("Over-sampling")
```

- **Compréhension du Problème :** Assurez-vous de bien comprendre la nature de votre problème de déséquilibre de classes. Parfois, d'autres méthodes, telles que la sous-échantillonnage, les méthodes basées sur l'ennemi (adversarial sampling), ou l'utilisation de poids de classe dans l'entraînement du modèle, peuvent également être considérées en fonction du contexte.
- **Validation Croisée :** Lorsque vous utilisez des techniques de prétraitement des données pour gérer le déséquilibre de classes, assurez-vous de les appliquer seulement sur l'ensemble d'entraînement. Gardez l'ensemble de test non modifié pour évaluer la performance réelle du modèle.
- **Évaluation du Modèle :** Après avoir appliqué la technique de suréchantillonnage, évaluez votre modèle sur plusieurs métriques, telles

que la précision, le rappel, la F1-score, etc., pour comprendre l'impact de la gestion du déséquilibre sur la performance du modèle.

- **Comparaison avec d'Autres Techniques :** Expérimentez également avec d'autres techniques de gestion du déséquilibre de classes pour voir laquelle fonctionne le mieux pour votre cas d'utilisation spécifique.

2.2 Scaling (Normalizing/Standardizing) Gwen

- librairies : Scikit-learn + sous-librairies (pipeline / preprocessing / metrics / model_selection)
- Scaling : le scaling ou mise à l'échelle est la transformation des caractéristiques d'un ensemble de données de manière à les ramener à une échelle commune. C'est souvent nécessaire dans le processus de développement d'un modèle de ML car de nombreuses méthodes reposent sur la mesure des distances entre les points de données ou l'optimisation de certaines fonctions, et ces mesures peuvent être sensibles à l'échelle des caractéristiques.
  - Normalizing (min-max scaling) : redimensionne les valeurs des caractéristiques dans un intervalle spécifique, généralement entre 0 et 1, en utilisant la plage (min-max) des valeurs de la caractéristique.
  - https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df)
scaled_features = scaler.transform(df)
#Convert to table format - MinMaxScaler
df_MinMax = pd.DataFrame(data=scaled_features, columns=["A
```

  - Standardizing (Z-score scaling) : transforme les valeurs des caractéristiques de manière à avoir une moyenne nulle et un écart type de 1.
  - https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_X = sc_X.fit_transform(df)
#Convert to table format - StandardScaler
sc_X = pd.DataFrame(data=sc_X, columns=["Age", "Salary","P
sc_X
```

Soufiane dit d'utiliser plusieurs scaler, pas que 2

2.3 Choix des algorithmes

Sck-learn (linear_model, neighbours, tree, SVM, ensemble -incl random forest-, naive bayes, neural network)

- **XGBoost (XGB_classifier) Doro**

```
from xgboost import XGBClassifier

# Exemple d'initialisation du classificateur XGBoost
model = XGBClassifier(
    learning_rate=0.1,
    n_estimators=100,
    max_depth=3,
    min_child_weight=1,
    gamma=0,
    subsample=1,
    colsample_bytree=1,
    objective='binary:logistic',
    nthread=-1,
    scale_pos_weight=1,
    seed=42
)
```

'Voici une explication de certains paramètres clés et des possibilités lors de l'initialisation de XGBClassifier :

- learning_rate : Contrôle le taux de réduction de la taille des pas lors de chaque mise à jour. Des valeurs plus basses rendent l'algorithme plus robuste mais peuvent nécessiter plus d'itérations.

- n_estimators : Le nombre de tours ou d'arbres de boosting à construire. Augmenter le nombre d'estimateurs améliore généralement les performances du modèle, mais peut également conduire à un surajustement.
- max_depth : Profondeur maximale des arbres de décision. Des arbres plus profonds peuvent capturer des motifs plus complexes mais peuvent entraîner un surajustement.
- min_child_weight : Somme minimale du poids des instances (hessien) nécessaire dans un enfant. C'est un terme de régularisation qui aide à contrôler le surajustement.
- gamma : Perte minimale requise pour effectuer une partition supplémentaire sur un nœud terminal. Il offre une autre manière de contrôler la complexité de l'arbre.
- subsample : Fraction d'échantillons utilisés pour l'entraînement de chaque arbre. Des valeurs plus basses aident à prévenir le surajustement.
- colsample_bytree : Fraction de caractéristiques utilisées pour l'entraînement de chaque arbre. Il contrôle le nombre de caractéristiques échantillonnées de manière aléatoire pour chaque arbre.
- objective : Spécifie la tâche d'apprentissage et la fonction objective correspondante. Pour les problèmes de classification, 'binary:logistic' est couramment utilisé.
- nthread : Nombre de threads parallèles utilisés pour le calcul. En le réglant sur -1, tous les threads disponibles sont utilisés.
- scale_pos_weight : Contrôle l'équilibre entre les poids positifs et négatifs. Utile pour les problèmes de classes déséquilibrées.
- seed : Graine aléatoire pour la reproductibilité.

- **Light Gradient Boosted Machine (LightGBM): Doro**

Framework de boosting de gradient conçu pour un entraînement distribué et efficace. Il a été développé par Microsoft et est particulièrement reconnu pour ses performances élevées et son efficacité.

```
# Importation de LightGBM
import lightgbm as lgb

# Exemple de création et d'entraînement d'un classificateur
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'boosting_type': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9
}

train_data = lgb.Dataset(X_train, label=y_train)
valid_data = lgb.Dataset(X_valid, label=y_valid, reference=

model = lgb.train(params, train_data, num_boost_round=100,

# Exemple de réalisation de prédictions
predictions = model.predict(X_test)
```

- **SVM (Support Vector Machine): Doro**

```
# Importation de SVM
from sklearn.svm import SVC  # pour la classification

# Création d'un classificateur SVM
classifier = SVC(kernel='linear', C=1.0)
# Entraînement du modèle
classifier.fit(X_train, y_train)
# Prédictions
predictions = classifier.predict(X_test)


from sklearn.svm import SVR  # pour la régression
```

```
# Création d'un modèle SVM pour la régression
regressor = SVR(kernel='linear', C=1.0)
# Entraînement du modèle
regressor.fit(X_train, y_train)
# Prédictions
predictions = regressor.predict(X_test)
```

- **Neural Network:** Caro

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

```
from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_s
...
clf.fit(X, y)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), randc
              solver='lbfgs')
```

Lors de l'utilisation du MLPClassifier de scikit-learn (sklearn.neural_network), il est important de prendre en compte plusieurs paramètres clés pour ajuster le modèle à vos besoins spécifiques.

Les données sont stockées sous forme de vecteurs et avec Python, vous stockez ces vecteurs dans des tableaux. Ce qui est intéressant à propos des couches de réseaux neuronaux, c'est que les mêmes calculs peuvent extraire des informations de n'importe quel type de données. Cela signifie que peu importe si vous utilisez des données d'image ou des données de texte. Le processus permettant d'extraire des informations significatives et de former le modèle d'apprentissage profond est le même pour les deux scénarios.

Voici quelques-uns des paramètres les plus importants à considérer :

1. **hidden_layer_sizes** : C'est un paramètre crucial qui spécifie la taille des couches cachées du réseau. Vous pouvez fournir un tuple indiquant le nombre de neurones dans chaque couche cachée. Par exemple, `(100, 50)` signifie deux couches cachées avec 100 neurones dans la première couche et 50 neurones dans la seconde.

2. **activation** : Cela détermine la fonction d'activation utilisée par les neurones. Les options courantes incluent 'logistic' (fonction sigmoïde), 'tanh' (tangente hyperbolique) et 'relu' (Rectified Linear Unit).
3. **solver** : Il s'agit de l'algorithme d'optimisation utilisé pour ajuster les poids du réseau. 'adam' est généralement une bonne option, mais vous pouvez également utiliser 'sgd' (descente de gradient stochastique) ou 'lbfgs' (une méthode quasi-Newton).
4. **alpha** : C'est le terme de régularisation qui contrôle la pénalité ajoutée aux poids du réseau pour éviter le surajustement.
5. **learning_rate** : Il détermine la taille des pas pris lors de la mise à jour des poids du réseau. Vous pouvez choisir entre 'constant', 'invscaling' et 'adaptive'.
6. **max_iter** : C'est le nombre maximum d'itérations (époques) pour lesquelles le solveur doit converger. Il est important de surveiller cela pour éviter le surajustement.
7. **batch_size** : La taille des mini-lots utilisée lors de l'optimisation. Vous pouvez spécifier 'auto' pour laisser le solveur choisir la taille du lot.
8. **learning_rate_init** : Il s'agit du taux d'apprentissage initial, qui est la taille du pas lors de la première mise à jour des poids.
9. **tol** : Tolérance pour la convergence. Si la perte n'améliore pas de plus que tol pendant deux époques consécutives, l'optimisation s'arrête.
10. **random_state** : Il permet de fixer une graine pour la reproductibilité des résultats.

- **Naïve Bayes: Gaussian Naive Bayes¶**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, t
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points
```

```
...       % (X_test.shape[0], (y_test != y_pred).sum()))
Number of mislabeled points out of a total 75 points : 4
```

- **CatBoost: CatBoost_classifier: Doro**

```
# Importation des bibliothèques nécessaires
import catboost
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Chargement des données (remplacez cela par vos propres do
# X représente les caractéristiques, y représente les étiqu
# Assurez-vous que vos caractéristiques catégorielles ne né
X_train, X_test, y_train, y_test = train_test_split(X, y, t

# Initialisation du modèle CatBoostClassifier
model = CatBoostClassifier(iterations=500, depth=10, learni

# Entraînement du modèle
model.fit(X_train, y_train, cat_features=categorical_featur

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

# Évaluation de la précision
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

### Autres paramètres disponibles :
#https://catboost.ai/en/docs/concepts/python-reference_catb
```

- **RandomForest:** c’est un algorithme d'apprentissage automatique qui appartient à la famille des méthodes ensemblistes. Il construit plusieurs arbres de décision lors de l'entraînement et les combine pour obtenir des prédictions plus robustes et générales. Chaque arbre est formé sur un sous-ensemble aléatoire des données et des caractéristiques, ce qui contribue à réduire le surajustement et à améliorer la performance

prédictive. Random Forest offre une puissante capacité de prédiction, résiste bien au surajustement grâce à la construction d'arbres aléatoires, et permet l'évaluation de l'importance des caractéristiques.

2.4 Sélection des variables explicatives et dépendantes

2.5 Entrainement des modèles

2.6 Test des modèles

2.7 Comparaison des modèles par rapport au choix du scaler et algorithme en utilisant pls métriques

Pour chaque scaler PUIS pour chaque classifier

```
from sklearn.metrics import confusion_matrix, accuracy_score,
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt

1. Matrice de Confusion
conf_matrix = confusion_matrix(y_true, y_pred)
print("Matrice de Confusion:")
print(conf_matrix)

2. Précision
accuracy = accuracy_score(y_true, y_pred)
print(f"Précision: {accuracy:.2f}")

3.
```

2.8 Stockage du meilleur modèle pour chaque maladie

2.9 Création du prototype du programme de test des modèles par variable des maladies

```
scaler_list= [Normalizer(), StandardScaler(), MinMaxScaler(),
classifier_list= [ RandomForestClassifier(), LogisticRegressi
                  MLPClassifier(), XGBClassifier(), LGBMClass
                  CatBoostClassifier(), GaussianNB()]

results= []



for scaler in scaler_list:
    for classifier in classifier_list:
        pipe = Pipeline([
        ('scaler', scaler),
        ('classifier', classifier)
        ])
        cross= cross_val_score(pipe, Xscaled, yscaled, cv= 5,
        y_pred = cross_val_predict(pipe, Xscaled, yscaled, cv=
        Accuracy= accuracy_score(yscaled, y_pred),
        Precision= precision_score(yscaled, y_pred),
        Recall= recall_score(yscaled, y_pred),
        F1_Score= f1_score(yscaled, y_pred),
        ROC_AUC= roc_auc_score(yscaled, y_pred)
        results.append((classifier.__class__.__name__, scaler
                        Accuracy, Precision, Recall, F1_Score
```

PROCESS FINAL

1. encoding - si besoin
2. oversampling - si besoin
3. pipepline
4. choix du modèle (scaler + algo) en fonction des scores (recall)
5. train test sur dataset d'origine (sur données non resamplées)

```
fit modele choisi

x_train, x_test, y_train, y_test = train_test_split(X_resampl

model_pipeline = Pipeline([
    ('scaler',MaxAbsScaler()),
    ('classifier',ExtraTreesClassifier())
])
model_pipeline.fit(x_train,y_train)
y_pred = model_pipeline.predict(x_test)

print(classification_report(y_test, y_pred))
```

6. possibilité de refaire le fit sur données resamplées mais pas obligatoire

Récap des scalers et modèles

Breast Cancer: Power Transformer et CatBoost

	Classifier	Scaler	Accuracy	Recall	Precision	F1-Score	ROC_AUC
46	CatBoostClassifier	PowerTransformer	(0.9709302325581395,)	(0.9850299401197605,)	(0.9563953488372093,)	(0.9705014749262537,)	0.970930
22	CatBoostClassifier	MinMaxScaler	(0.9709302325581395,)	(0.9850299401197605,)	(0.9563953488372093,)	(0.9705014749262537,)	0.970930
30	CatBoostClassifier	MaxAbsScaler	(0.9709302325581395,)	(0.9850299401197605,)	(0.9563953488372093,)	(0.9705014749262537,)	0.970930
14	CatBoostClassifier	StandardScaler	(0.9709302325581395,)	(0.9850299401197605,)	(0.9563953488372093,)	(0.9705014749262537,)	0.970930
38	CatBoostClassifier	RobustScaler	(0.9709302325581395,)	(0.9850299401197605,)	(0.9563953488372093,)	(0.9705014749262537,)	0.970930
36	XGBClassifier	RobustScaler	(0.9651162790697675,)	(0.9819277108433735,)	(0.9476744186046512,)	(0.9644970414201183,)	0.965116
12	XGBClassifier	StandardScaler	(0.9651162790697675,)	(0.9819277108433735,)	(0.9476744186046512,)	(0.9644970414201183,)	0.965116
28	XGBClassifier	MaxAbsScaler	(0.9651162790697675,)	(0.9819277108433735,)	(0.9476744186046512,)	(0.9644970414201183,)	0.965116
20	XGBClassifier	MinMaxScaler	(0.9651162790697675,)	(0.9819277108433735,)	(0.9476744186046512,)	(0.9644970414201183,)	0.965116
44	XGBClassifier	PowerTransformer	(0.9651162790697675,)	(0.9819277108433735,)	(0.9476744186046512,)	(0.9644970414201183,)	0.965116
29	LGBMClassifier	MaxAbsScaler	(0.9680232558139535,)	(0.9791666666666666,)	(0.9563953488372093,)	(0.9676470588235294,)	0.968023

	0 PREDICTED	1 PREDICTED
0 ACTUAL	75	4
1 ACTUAL	3	90

Heart disease : Power Transformer et GaussianNB

0 == malade
1 == sain

[24]:

	Classifier	Scaler	Accuracy	Recall	Precision	F1-Score	ROC_AUC
47	GaussianNB	PowerTransformer	(0.8344370860927153,)	(0.8475609756097561,)	(0.8475609756097561,)	(0.8475609756097561,)	0.833201
8	RandomForestClassifier	StandardScaler	(0.8344370860927153,)	(0.8433734939759037,)	(0.8536585365853658,)	(0.8484848484848484,)	0.832626
11	MLPClassifier	StandardScaler	(0.8443708609271523,)	(0.8421052631578947,)	(0.8780487804878049,)	(0.8597014925373135,)	0.841198
32	RandomForestClassifier	RobustScaler	(0.8278145695364238,)	(0.8373493975903614,)	(0.8475609756097561,)	(0.8424242424242424,)	0.825954
43	MLPClassifier	PowerTransformer	(0.8410596026490066,)	(0.8372093023255814,)	(0.8780487804878049,)	(0.8571428571428572,)	0.837575
0	RandomForestClassifier	Normalizer	(0.8410596026490066,)	(0.8372093023255814,)	(0.8780487804878049,)	(0.8571428571428572,)	0.837575
40	RandomForestClassifier	PowerTransformer	(0.8178807947019867,)	(0.8343558282208589,)	(0.8292682926829268,)	(0.8318042813455657,)	0.816808
42	SVC	PowerTransformer	(0.8410596026490066,)	(0.8333333333333334,)	(0.8841463414634146,)	(0.8579881656804734,)	0.837001
16	RandomForestClassifier	MinMaxScaler	(0.8278145695364238,)	(0.8333333333333334,)	(0.8536585365853658,)	(0.8433734939759037,)	0.825380
19	MLPClassifier	MinMaxScaler	(0.8377483443708609,)	(0.8323699421965318,)	(0.8780487804878049,)	(0.85459940652819,)	0.833952

	0 PREDICTED	1 PREDICTED
0 ACTUAL	22	8
1 ACTUAL	4	42

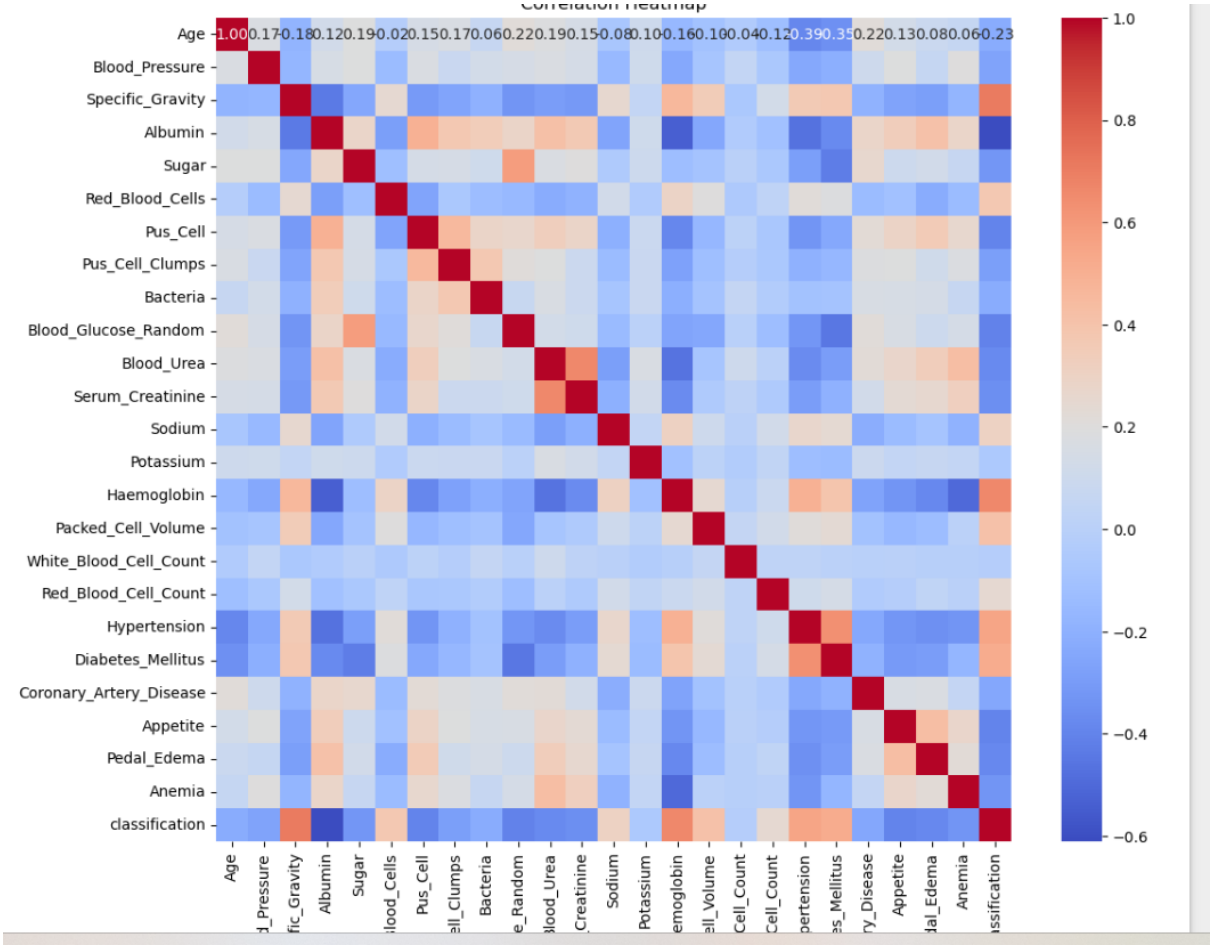
CKD : Standard Scaler et RandomForest

0 == malade
1 == sain

[20]:

	Classifier	Scaler	Accuracy	Recall	Precision	F1-Score	ROC_AUC
8	RandomForestClassifier	StandardScaler	(0.992,)	(0.992,)	(0.992,)	(0.992,)	0.992
16	RandomForestClassifier	MinMaxScaler	(0.99,)	(0.9919678714859438,)	(0.988,)	(0.9899799599198397,)	0.990
24	RandomForestClassifier	MaxAbsScaler	(0.99,)	(0.9880478087649402,)	(0.992,)	(0.9900199600798404,)	0.990
40	RandomForestClassifier	PowerTransformer	(0.99,)	(0.9880478087649402,)	(0.992,)	(0.9900199600798404,)	0.990
32	RandomForestClassifier	RobustScaler	(0.99,)	(0.9880478087649402,)	(0.992,)	(0.9900199600798404,)	0.990
37	LGBMClassifier	RobustScaler	(0.99,)	(0.9841897233201581,)	(0.996,)	(0.9900596421471173,)	0.990
29	LGBMClassifier	MaxAbsScaler	(0.99,)	(0.9841897233201581,)	(0.996,)	(0.9900596421471173,)	0.990

	0 PREDICTED	1 PREDICTED
0 ACTUAL	62	0
1 ACTUAL	0	63



Diabete: Random Forest SANS classifier

0 == sain

1== malade

[23]:

	Classifier	Scaler	Accuracy	Recall	Precision	F1-Score	ROC_AUC
8	RandomForestClassifier	StandardScaler	(0.806,)	(0.7703180212014135,)	(0.872,)	(0.8180112570356473,)	0.806
5	LGBMClassifier	Normalizer	(0.79,)	(0.7665441176470589,)	(0.834,)	(0.7988505747126436,)	0.790
24	RandomForestClassifier	MaxAbsScaler	(0.799,)	(0.7646017699115044,)	(0.864,)	(0.8112676056338028,)	0.799
16	RandomForestClassifier	MinMaxScaler	(0.803,)	(0.7643979057591623,)	(0.876,)	(0.8164026095060578,)	0.803
32	RandomForestClassifier	RobustScaler	(0.796,)	(0.7587412587412588,)	(0.868,)	(0.8097014925373135,)	0.796
40	RandomForestClassifier	PowerTransformer	(0.796,)	(0.7534246575342466,)	(0.88,)	(0.8118081180811809,)	0.796

	0 PREDICTED	1 PREDICTED
0 ACTUAL	93	21
1 ACTUAL	15	121

Liver Diseases: Random Forest et MaxAbsScaler

1 == malade
0/ 2 == sain

[19]:

	Classifier	Scaler	Accuracy	Recall	Precision	F1-Score	ROC_AUC
24	RandomForestClassifier	MaxAbsScaler	(0.833743842364532,)	(0.9301587301587302,)	(0.7216748768472906,)	(0.8127600554785019,)	0.833744
16	RandomForestClassifier	MinMaxScaler	(0.8411330049261084,)	(0.9287925696594427,)	(0.7389162561576355,)	(0.823045267489712,)	0.841133
8	RandomForestClassifier	StandardScaler	(0.8251231527093597,)	(0.9285714285714286,)	(0.7044334975369458,)	(0.8011204481792716,)	0.825123
32	RandomForestClassifier	RobustScaler	(0.8275862068965517,)	(0.9262820512820513,)	(0.7118226600985221,)	(0.8050139275766017,)	0.827586
44	XGBClassifier	PowerTransformer	(0.8374384236453202,)	(0.9254658385093167,)	(0.7339901477832512,)	(0.8186813186813188,)	0.837438
36	XGBClassifier	RobustScaler	(0.8374384236453202,)	(0.9254658385093167,)	(0.7339901477832512,)	(0.8186813186813188,)	0.837438

	1 PREDICTED	2 PREDICTED
1 ACTUAL	84	24
2 ACTUAL	12	83