

# Rapport n°2: Machine Learning

Nous passons maintenant à l'étape des tests de machine learning. L'objectif est d'essayer différents modèles, différentes configurations, dans le but d'obtenir les meilleures prédictions possible.

Rappel: Notre variable gravité est séparée en trois catégories:

- Blessé léger
- Blessé hospitalisé
- Tué

## Quelle est la classe dont nous souhaitons optimiser la prédiction?

Pour rappel, notre objectif est de prédire les gravité de l'accident afin de mobiliser les secours adéquats. **Nous souhaitons donc prédire au mieux la classe des blessés hospitalisés.** En effet, pour les blessés légers, pas d'urgence de secours, et malheureusement, pour les tués non plus.

## Quels modèles allons-nous tester?

Il s'agit d'un problème de classification. **Nous testons donc différents modèles de classification:** Random Forest, LogisticRegression, XGBoost, KNN. Notons que les classes sont très déséquilibrées, nous testons donc l'oversampling et l'undersampling dans tous les cas.

Nous avons également pensé à prédire le nombre de blessés hospitalisés (par département par exemple) par mois en utilisant une série temporelle. Ainsi, les services de secours pourraient optimiser leur planning et leurs ressources. Hélas le résidu présente de trop grandes variations. Nous présenterons ces essais brièvement, et resterons finalement sur les modèles de classification précédemment cités.

## Après avoir testé nos modèles, comment évaluer leur performance? Que cherchons nous à optimiser?

Nous désirons que notre modèle prédise au mieux les vrais positifs pour avoir suffisamment de secours pour cette classe, et qu'il indique le moins de faux positifs pour ne pas mobiliser de secours inutilement. **Notre but est donc d'optimiser le Recall (bonne prédiction des vrais positifs), et le F1\_Score( peu de faux positifs).**

Notre travail est divisé en 5 étapes:

- Préparer un dataframe `df_machine_learning` avec les variables que l'on garde pour les modèles.
- Tester les différents modèles de manière très simple.
- Tester les mêmes modèles avec les 6 variables reconnues comme les plus importantes par le modèle RandomForest.
- Tester les différents modèles avec de l'undersampling et de l'oversampling
- Tenter d'améliorer les performances de notre modèle, selon les axes définis (Recall et F1 score des blessés hospitalisés).

Pour ce faire, nous pensons à deux moyens: - Créer une métrique `make_scorer` qui optimise le F1\_Score (ou le Recall) de la classe blessés hospitalisés. - Tester plusieurs pondérations des classes, et évaluer le recall (ou F1\_Score) selon ces différentes pondérations, et conserver la meilleure. - Refaire tourner les même modèles, mais cette fois, avec uniquement deux classes: Non\_Urgent (qui regroupe blessés légers et tués), et Urgent (blessés hospitalisés), et voir si les résultats sont meilleurs.

A noter: certains modèles n'ont pas pu tourner, le noyau a planté. Nous ne sommes pas équipés d'ordinateurs suffisamment puissants. C'est le cas de la LogisticRegression, et d'autre modèles dont nous n'avons pas fourni les résultats.

# Tableau récapitulatif des résultats obtenus

## Modèle 3 classes Random Forest

RANDOM FOREST	Best parameters	Accuracy	Precision			Recall			F1_Score		
			Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué
Simple		0.69	0.76	0.53	0.27	0.85	0.46	0.09	0.80	0.49	0.14
Simple avec 6 variables les plus importantes		0.69	0.74	0.54	0.19	0.88	0.41	0.02	0.80	0.47	0.03
Oversampling		0.68	0.77	0.52	0.25	0.83	0.47	0.16	0.80	0.50	0.20
Undersampling		0.58	0.82	0.83	0.16	0.67	0.39	0.58	0.74	0.41	0.25
Grid search avec class_weight = balanced	{'classifier__class_weight': None, 'classifier__max_depth': 20, 'classifier__n_estimators': 100}	0.72	0.76	0.58	0.48	0.89	0.47	0.03	0.82	0.52	0.06
Métrique pondérée pour optimiser le Recall de la classe hospitalisé	{0: 1, 1: 2, 2: 1}, n_estimator = 50, max_depth = 10	0.70	0.80	0.52	0.50	0.79	0.62	0.00	0.80	0.57	0.00
Métrique pondérée pour optimiser le F1_SCORE de la classe hospitalisés	{0: 1, 1: 2, 2: 1}, n_estimator = 50, max_depth = 10	0.70	0.80	0.52	0.50	0.79	0.62	0.00	0.80	0.57	0.00
Métrique pondérée pour meilleur RECALL de la classe hospitalisés	{0: 1, 1: 2, 2: 1}, n_estimator = 50, max_depth = 10	0.70	0.80	0.52	0.50	0.79	0.62	0.00	0.80	0.57	0.00

## Modèle 3 classes Régression Logistique

LogisticReg	Best parameters	Accuracy	Precision			Recall			F1_Score		
			Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué
Simple		0.70	0.75	0.57	0.26	0.90	0.43	0.01	0.81	0.49	0.01
Oversampling		0.67	0.82	0.45	0.15	0.70	0.35	0.61	0.75	0.39	0.25
Undersampling		0.59	0.82	0.45	0.15	0.70	0.34	0.60	0.75	0.39	0.25
Grid search	{'classifier__C': 0.01, 'classifier__solver': 'liblinear'}	0.68	0.78	0.53	0.22	0.82	0.44	0.27	0.80	0.48	0.24

## Modèle 3 classes KNN

KNN	Best parameters	Accuracy	Precision			Recall			F1_Score		
			Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué
Simple											
Oversampling											
Undersampling											
Grid search	{'classifier__n_neighbors': 9, 'classifier__weights': 'uniform'}	0.69	0.74	0.54	0.35	0.87	0.43	0.06	0.80	0.48	0.11
Grid search avec scoring personnalisé pour optimiser F1_score classe hospitalisés		0.68	0.75	0.52	0.28	0.85	0.44	0.06	0.79	0.48	0.10

## Modèle 3 classes XGBoost

XGBOOST	Best parameters	Accuracy	Precision			Recall			F1_Score		
			Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué	Léger	Hospitalisé	Tué
Simple		0.63	0.84	0.50	0.19	0.73	0.43	0.61	0.78	0.46	0.29
Oversampling		0.72	0.77	0.59	0.39	0.89	0.46	0.14	0.82	0.52	0.2
Undersampling											
Grid search	{'classifier__max_depth': 6, 'classifier__n_estimators': 100, 'classifier__scale_pos_weight': 1}	0.72	0.77	0.58	0.46	0.89	0.49	0.07	0.83	0.53	0.12
Grid search avec scoring personnalisé pour optimiser F1_score classe hospitalisés		0.72	0.77	0.59	0.39	0.89	0.47	0.13	0.82	0.52	0.20

## Modèle 2 classes Random Forest

classes_RandomFore	Best parameters	Accuracy	Precision		Recall		F1_Score	
			Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)
Simple		0.72	0.77	0.54	0.86	0.39	0.81	0.45
Oversampling		0.71	0.78	0.53	0.83	0.44	0.80	0.48
Undersampling		0.67	0.83	0.47	0.67	0.67	0.74	0.55
Grid search	{'classifier__class_weight': 'balanced', 'classifier__max_depth': 20, 'classifier__n_estimators': 200}	0.72	0.82	0.53	0.77	0.60	0.80	0.56
Grid search avec scoring personnalisé pour optimiser recall classe Urgent		0.70	0.83	0.51	0.73	0.65	0.78	0.57
Grid search avec scoring personnalisé pour optimiser F1_score classe hospitalisés								
Métrique pondérée pour meilleur recall de la classe urgent	{0: 1, 1: 2}	0.72	0.82	0.53	0.77	0.60	0.79	0.56

## Modèle 2 classes Régression Logistique

2_classes_LogisticReg	Best parameters	Accuracy	Precision		Recall		F1_Score	
			Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)
Simple		0.73	0.76	0.60	0.91	0.32	0.83	0.41
Oversampling		0.68	0.83	0.47	0.68	0.68	0.75	0.56
Undersampling		0.68	0.83	0.48	0.68	0.67	0.75	0.56

## Modèle 2 classes KNN

2_classes_KNN	Best parameters	Accuracy	Precision		Recall		F1_Score	
			Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)
Simple								
Oversampling								
Undersampling								
Grid search								
Grid search avec scoring personnalisé pour optimiser recall classe hospitalisés								
Grid search avec scoring personnalisé pour optimiser Recall classe hospitalisés		0.70	0.77	0.49	0.81	0.42	0.79	0.45

## Modèle 2 classes XGBoost

2_classes_XGBOOST	Best parameters	Accuracy	Precision		Recall		F1_Score	
			Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)	Non urgent (léger/tué)	Urgent (hospitalisé)
Simple		0.70	0.84	0.50	0.70	0.70	0.77	0.59
Oversampling		0.74	0.79	0.58	0.85	0.47	0.82	0.52
Undersampling								
Grid search								
Grid search avec scoring personnalisé pour optimiser recall classe hospitalisés								
Grid search avec scoring personnalisé pour optimiser Recall classe hospitalisés		0.70	0.77	0.49	0.81	0.42	0.79	0.45
Grid search avec scoring personnalisé pour optimiser F1_score classe hospitalisés								
Métrique pondérée pour meilleur F1_score de la classe hospitalisés								

## Conclusion:

Parmi tous les modèles et configurations testés, le modèle Random Forest est le plus performant. Il est optimal lorsque l'on utilise les paramètres suivants:

- {'classifier\_\_max\_depth': 10, 'classifier\_\_n\_estimators': 50}
- Pondération des classes pour optimiser la Recall (et le F1\_score): {0: 1, 1: 2, 2: 1}

Dans ce cas, le recall s'élève à 0.62 et le F1\_Score à 0.57 pour la classe blessés hospitalisés.

On constate que les modèles avec uniquement 2 classes pour la variable cible ne sont pas beaucoup plus performants que les modèles avec 3 classes pour la variable cible. On s'imaginait un plus gros écart.

Le meilleur modèle avec une cible à deux classes est le XGBoost avec l'argument `class_weight = balanced` pour gérer le déséquilibre des classes. Dans ce cas, le Recall s'élève à 0.70 et le F1\_score à 0.59 pour la classe urgent.

---

Vous trouverez ci-dessous tous les codes utilisés pour ces essais:

- I. Elaboration d'un dataframe avec les variables sélectionnées , nommé `df_machine learning`.
- II. Travail sur les séries temporelles
- III. Modèle 3 classes Random Forest
- IV. Modèle 3 classes Régression Logistique
- V. Modèle 3 classes KNN
- VI. Modèle 3 classes XGBoost
- VII. Modèle 2 classes Random Forest
- VIII. Modèle 2 classes Régression Logistique
- IX. Modèle 2 classes KNN
- X. Modèle 2 classes XGBoost

# I. Elaboration d'un dataframe avec les variables sélectionnées , nommé df\_machine learning.

```
from google.colab import drive
import pandas as pd
import os
from io import StringIO
# Monter Google Drive
drive.mount('/content/drive', force_remount= True) #force_remount = True pe

file_path = '/content/drive/My Drive/Datascientest/Projet_accidents/Dataset/2
df = pd.read_csv(file_path)
df.head(10)
df.info()
df.gravité_accident.value_counts()
```





Mounted at /content/drive

<ipython-input-1-21e3c8667d5e>:9: DtypeWarning: Columns (12) have mixed types. Specify dtype option on import or set low\_memory=False.

```
df = pd.read_csv(file_path)
```

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 273226 entries, 0 to 273225

Data columns (total 60 columns):

#	Column	Non-Null Count	Dtype
0	Num_Acc	273226 non-null	int64
1	jour	273226 non-null	int64
2	mois	273226 non-null	int64
3	an	273226 non-null	int64
4	hrmn	273226 non-null	object
5	dep	273226 non-null	object
6	agg	273226 non-null	int64
7	int	273226 non-null	int64
8	atm	273226 non-null	int64
9	col	273226 non-null	int64
10	lat	273226 non-null	object
11	long	273226 non-null	object
12	nbv	273226 non-null	object
13	vma	273226 non-null	int64
14	nationale_departementale_communale	273226 non-null	int64
15	autoroute	273226 non-null	int64
16	autre_route	273226 non-null	int64
17	sens_unique	273226 non-null	int64
18	bidirectionnel	273226 non-null	int64
19	route_seche	273226 non-null	int64
20	route_mouillee_enneigee	273226 non-null	int64
21	etat_route_autre	273226 non-null	int64
22	usager_count	273226 non-null	int64
23	indemne	273226 non-null	int64
24	tué	273226 non-null	int64
25	blessé_hospitalisé	273226 non-null	int64

26	blessé_léger	273226 non-null int64
27	total_sans_secu	273226 non-null int64
28	total_ceinture	273226 non-null int64
29	total_casque	273226 non-null int64
30	total_secu_enfant	273226 non-null int64
31	total_gilet	273226 non-null int64
32	total_airbag	273226 non-null int64
33	total_gants	273226 non-null int64
34	total_gants_airbag	273226 non-null int64
35	total_autre	273226 non-null int64
36	place_conducteur	273226 non-null int64
37	pax_AV	273226 non-null int64
38	pax_AR	273226 non-null int64
39	pax_Milieu	273226 non-null int64
40	place_pieton	273226 non-null int64
41	homme	273226 non-null int64
42	femme	273226 non-null int64
43	0-17	273226 non-null int64
44	18-60	273226 non-null int64
45	61-95	273226 non-null int64
46	obstacle_fixe	273226 non-null int64
47	obstacle_mobile	273226 non-null int64
48	aucun_choc	273226 non-null int64
49	choc_AV	273226 non-null int64
50	choc_AR	273226 non-null int64
51	choc_cote	273226 non-null int64
52	choc_tonneaux	273226 non-null int64
53	VL_VU	273226 non-null int64
54	2roues_3roues_quad	273226 non-null int64
55	PL	273226 non-null int64
56	bus_car	273226 non-null int64
57	velo_trott_edp	273226 non-null int64
58	nbr_veh	273226 non-null int64
59	gravité_accident	273226 non-null int64

dtypes: int64(55), object(5)  
memory usage: 125.1+ MB

	count
gravité_accident	
2	175525
3	82229
4	15472

dtype: int64

```
# Nous conservons uniquement l'heure de l'accident
```

```
df['date'] = pd.to_datetime(df['jour'].astype('str') + '/' + df['mois'].astype('str') + ' ' + df['heure']).dt.hour
```

# Modification du type de la variable gravité\_accident (cible), en variable catégorielle

```
df['gravité_accident'] = df['gravité_accident'].astype('category')
```

Nettoyage colonne nombre de voies (nbv), certaines valeurs n'étant pas numérique, et d'autres irréalistes (Les routes à 11 et 12 voies n'existent pas en France). On les remplace par le mode de la variable.

```
mode_value = df['nbv'].mode()[0]
```

```
# Remplacement des valeurs spécifiques
```

```
df['nbv'] = df['nbv'].replace([-1, 11, 12, '-1', '-1', 0], mode_value)
```

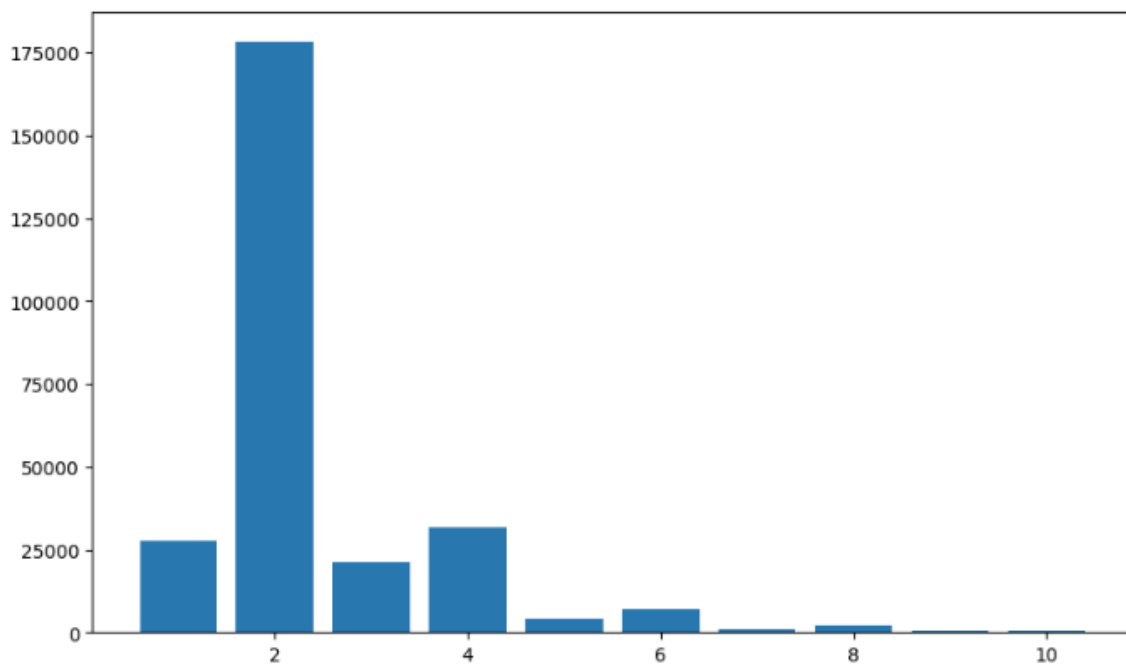
```
df['nbv'] = df['nbv'].replace(['#VALEURMULTI', '-1', '0', '#ERREUR', '11', '12'], mode_value)
```

```
df['nbv'] = df['nbv'].replace({'2': 2, '8': 8, '6': 6, '4': 4, '5': 5, '7': 7, '3': 3, '1': 1, '0': 0})
```

	count
nbv	
2	178149
4	31627
1	27445
3	21215
6	6864
5	4219
8	2125
7	755
10	510
9	317

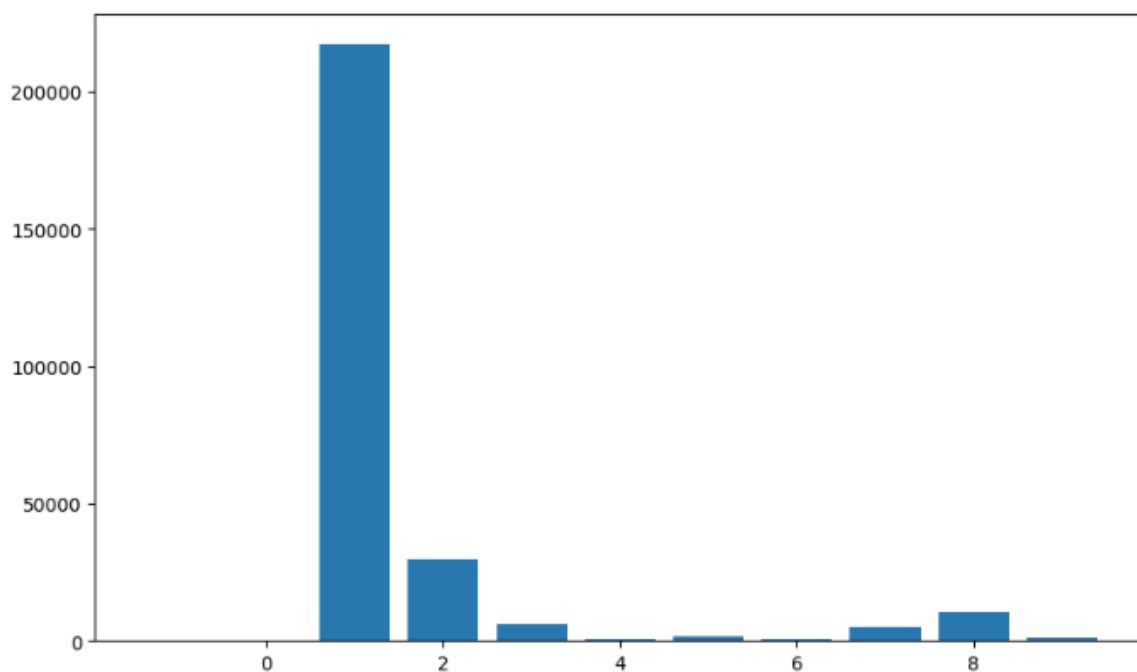
dtype: int64

```
import matplotlib.pyplot as plt
nbv_counts = df.nbv.value_counts()
plt.figure(figsize=(10, 6))
plt.bar(nbv_counts.index, nbv_counts.values)
plt.show();
```



Nettoyage colonne atmosphère (atm): on les regroupe en 4 catégories: 'Temps normal', 'temps couvert', 'temps pluvieux', 'autre'

```
df.atm.value_counts()
import matplotlib.pyplot as plt
atm_counts = df.atm.value_counts()
plt.figure(figsize=(10, 6))
plt.bar(atm_counts.index, atm_counts.values)
plt.show();
```



```
def regrouper(val):
    if val in [1]:
        return 'temps_normal'
    elif val in [2, 3]:
        return 'Temps_pluvieux'
    elif val in [8]:
        return 'Temps_couvert'
    else:
        return 'Autre'
```

```
df['atm'] = df['atm'].apply(regrouper)
df.atm.value_counts()
```

```

              count
atm
temps_normal    217143
Temps_pluvieux   36030
Temps_couvert   10470
Autre            9583
dtype: int64
```

```
# Enregistrement du df pour le machine learning
df.to_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/201
```

## II. Travail sur la série temporelle

```
# Préparation d'un df pour la série temporelle
```

```
from google.colab import drive
import pandas as pd
import os
from io import StringIO
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Monter Google Drive
```

```
drive.mount('/content/drive', force_remount= True) #force_remount = True pe
```

```
file_path = '/content/drive/My Drive/Datascientest/Projet_accidents/Dataset/2
df_total_final=pd.read_csv("/content/drive/MyDrive/Datascientest/Projet_acci
df_temp = df_total_final.copy()
df_temp['date']= pd.to_datetime(df_temp['jour'].astype('str')+'/' +df_temp['mo
df_temp['an_mois_jour'] = df_temp['date'].dt.strftime('%Y-%m-%d')
```

```
df_temp = df_temp[['an_mois_jour', 'blessé_léger', 'blessé_hospitalisé', 'tué', 'd

print(df_temp.head())
# On enregistre le df_total_final dans le drive partagé
df_temp.to_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Datas

df_temp.dep.value_counts()
```

```
Mounted at /content/drive
<ipython-input-9-dde33cbf1a44>:15: DtypeWarning: Columns (12) have mixed types. Specify dtype option on import or set low_memory=False.
df_total_final=pd.read_csv("/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_total_final.csv")
  an_mois_jour  blessé_léger  blessé_hospitalisé  tué  dep
0  2019-11-30             2                   0    0  93
1  2019-11-30             1                   0    0  93
2  2019-11-28             2                   0    0  92
3  2019-11-30             1                   0    0  94
4  2019-11-30             1                   0    0  94

count
dep
75    25239
93    13955
13    12353
92    11882
94    11820
...      ...
9      124
8       91
977     65
986     56
975     17
116 rows x 1 columns

dtype: int64
```

#Il y a 25239 accidents dans le département 75, 13955 dans le 93, 12353 dans le 13, 11882 dans le 92, 11820 dans le 94, 124 dans le 9, 91 dans le 8, 65 dans le 977, 56 dans le 986, 17 dans le 975.

#Nous allons tester des prédictions par analyse temporelle, sur le département 75.

#Cette analyse peut s'appliquer à tous les départements (s'ils ont suffisamment d'accidents).

# Filtrage pour le département 75

```
df_temp = pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_total_final.csv')
df_temp_75 = df_temp[df_temp['dep'] == '75']
df_temp_75.head()
```

	blessé_léger	blessé_hospitalisé	tué	dep
an_mois_jour				
2019-10-24	1	0	0	75
2019-10-23	3	0	0	75
2019-10-23	1	0	0	75
2019-10-23	1	0	0	75
2019-10-24	1	0	0	75

# Transformation en série temporelle

```
df_temps_75 = df_temp_75.copy()
df_temp_75 = df_temp_75.drop('dep', axis = 1)
df_temp_75_resampled = df_temp_75.resample('D').sum().resample('M').sum()
df_temp_75_resampled.index
df_temp_75_resampled.head()
```

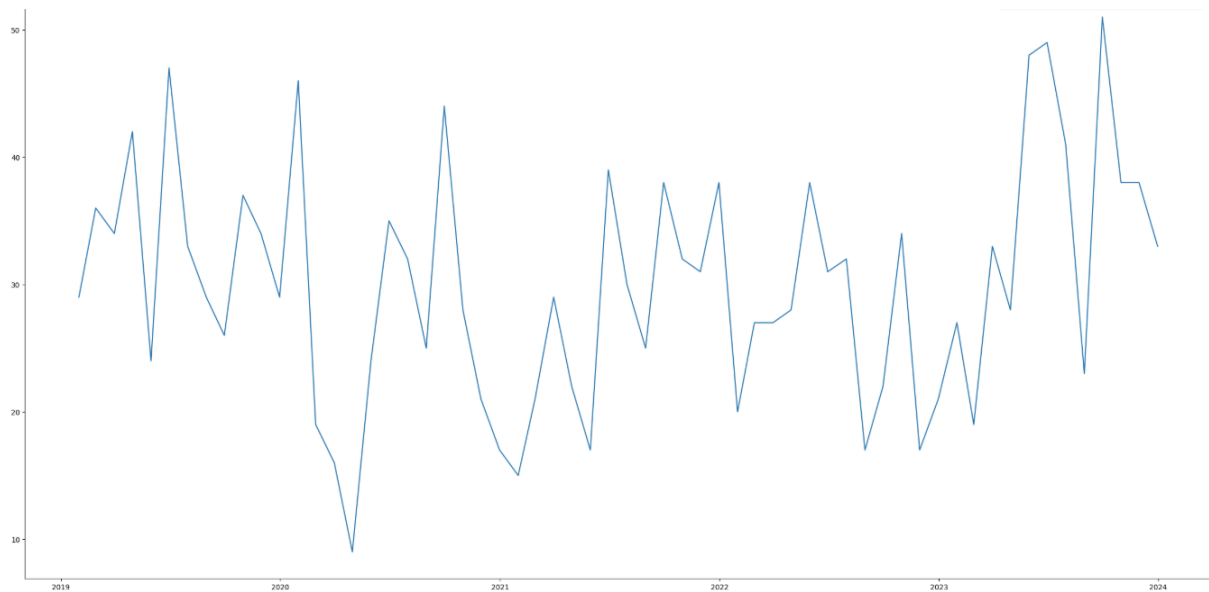
```
<ipython-input-11-ec64435788fb>:5: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
df_temp_75_resampled = df_temp_75.resample('D').sum().resample('M').sum()
```

	blessé_léger	blessé_hospitalisé	tué
an_mois_jour			
2019-01-31	472	29	1
2019-02-28	434	36	3
2019-03-31	495	34	0
2019-04-30	509	42	5
2019-05-31	488	24	2

# Etude approfondie des blessés hospitalisés en vue d'une prédiction

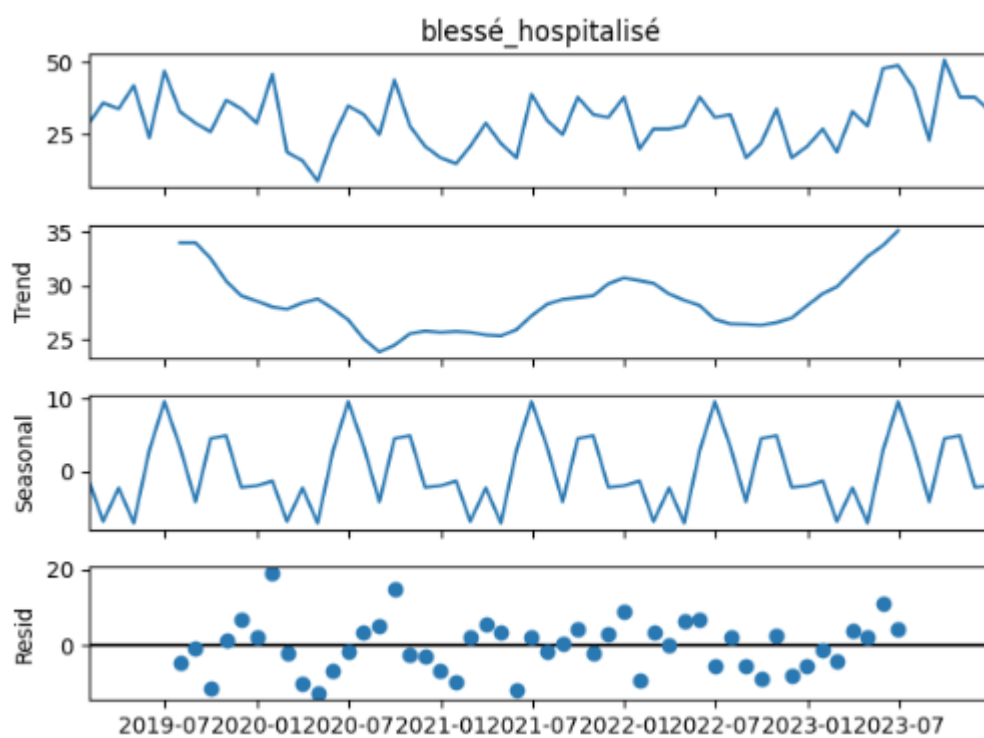
```
plt.figure(figsize = (30,15))
plt.plot('blessé_hospitalisé',data=df_temp_75_resampled)
plt.show()
```





```
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
variable_hospitalisé= seasonal_decompose(df_temp_75_resampled['blessé_hospitalisé'])
variable_hospitalisé.plot()
plt.show();
```



# On observe des résidus avec de grandes variations, mais une saisonnalité

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error
```

# hospitalisé

```
df_temp_75_resampled['blessé_hospitalisé_lag1'] = df_temp_75_resampled['blessé_hospitalisé'].shift(1)
df_temp_75_resampled.head()
```

	blessé_léger	blessé_hospitalisé	tué	blessé_hospitalisé_lag1
an_mois_jour				
2019-01-31	472	29	1	NaN
2019-02-28	434	36	3	29.0
2019-03-31	495	34	0	36.0
2019-04-30	509	42	5	34.0
2019-05-31	488	24	2	42.0

#Création de variables explicatives à partir de la série temporelle (moyenne mobile)

#Hospitalisé

```
df_temp_75_resampled['blessé_hospitalisé_MA_3months'] = df_temp_75_resampled['blessé_hospitalisé'].rolling(3).mean()
df_temp_75_resampled.head()
```

	blessé_léger	blessé_hospitalisé	tué	blessé_hospitalisé_lag1	blessé_hospitalisé_MA_3months
an_mois_jour					
2019-01-31	472	29	1	NaN	NaN
2019-02-28	434	36	3	29.0	NaN
2019-03-31	495	34	0	36.0	33.000000
2019-04-30	509	42	5	34.0	37.333333
2019-05-31	488	24	2	42.0	33.333333

#Création de variables explicatives à partir de la série temporelle (moyenne mobile)

#Hospitalisé

```
df_temp_75_resampled['blessé_hospitalisé_MA_3months'] = df_temp_75_resampled['blessé_hospitalisé'].rolling(3).mean()
df_temp_75_resampled.head()
```

```
# Suppression des Nans
```

```
df_temp_75_resampled= df_temp_75_resampled.dropna()
```

```
df_temp_75_resampled.head()
```

	blessé_léger	blessé_hospitalisé	tué	blessé_hospitalisé_lag1	blessé_hospitalisé_MA_3months
an_mois_jour					
2019-03-31	495	34	0	36.0	33.000000
2019-04-30	509	42	5	34.0	37.333333
2019-05-31	488	24	2	42.0	33.333333
2019-06-30	545	47	5	24.0	37.666667
2019-07-31	520	33	5	47.0	34.666667

```
# Normalisation des données
```

```
# Hospitalisé
```

```
y_hospitalisé = df_temp_75_resampled['blessé_hospitalisé']
```

```
X_hospitalisé = df_temp_75_resampled.drop('blessé_hospitalisé', axis = 1)
```

```
model= LinearRegression()
```

```
# Hospitalisé
```

```
X_hospitalisé_train = X_hospitalisé.iloc[: -24]
```

```
X_hospitalisé_test = X_hospitalisé.iloc[-24:]
```

```
y_hospitalisé_train = y_hospitalisé[:-24]
```

```
y_hospitalisé_test = y_hospitalisé.iloc[-24:]
```

```
model.fit(X_hospitalisé_train, y_hospitalisé_train)
```

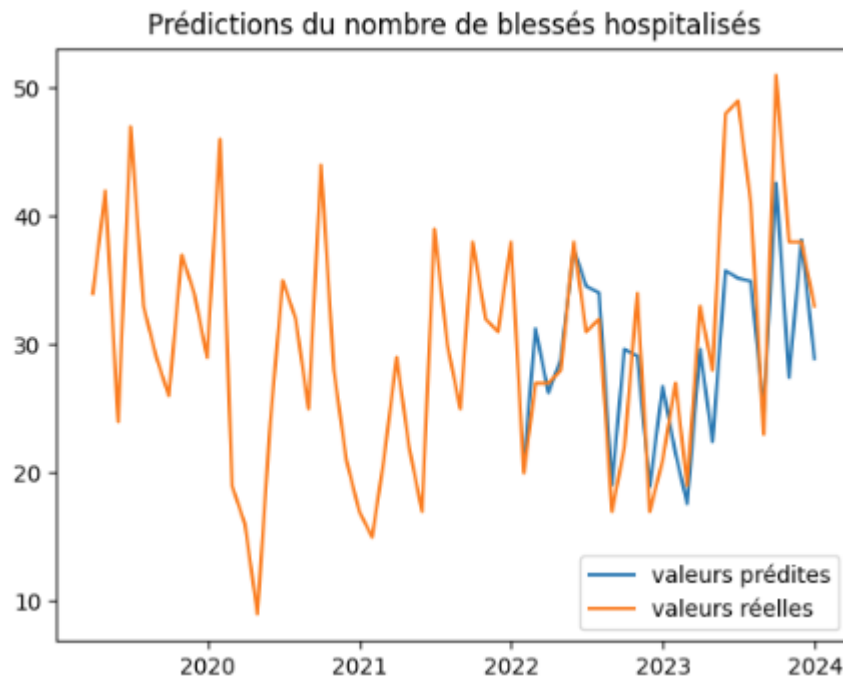
```
pred_test_hospitalisé = model.predict(X_hospitalisé_test)
```

```
print ('rmse', root_mean_squared_error(pred_test_hospitalisé, y_hospitalisé_te
```

```
plt.plot(y_hospitalisé_test.index, pred_test_hospitalisé, label = 'valeurs prédite
```

```
plt.plot(y_hospitalisé.index, y_hospitalisé, label = 'valeurs réelles')
```

```
plt.title('Prédictions du nombre de blessés hospitalisés')
plt.legend()
plt.show();
```



Nous décidons de ne pas poursuivre ce essais, puisque les résidus présentent de grandes variations, et que cette première tentative de prédiction est loin de la réalité.

### III. Modèle 3 classes Random Forest

```
from google.colab import drive
import pandas as pd
import os
from io import StringIO
# Monter Google Drive
drive.mount('/content/drive', force_remount= True) #force_remount = True pe
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Data:
df['gravité_accident'] = df['gravité_accident']-2
df.info()
```

```
# 0: blessé_léger
# 1: blessé_hospitalisé
# 2: tué
```

Import des bibliothèques nécessaires

```
import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as imPipeline
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import precision_score, recall_score, f1_score, confusion
import seaborn as sns
from sklearn.model_selection import GridSearchCV
```

## Modèle Random Forest simple

```
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int']
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
```

```

    return self

def transform(self, X):
    X = X.astype(float) # Assurer que les valeurs sont numériques
    X_sin = np.sin(2 * np.pi * X / self.period)
    X_cos = np.cos(2 * np.pi * X / self.period)
    return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
    'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
    '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))]

# Entraîner le modèle
pipeline.fit(X_train, y_train)

```

```

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accurac   du mod  le : {accuracy:.2f}")

# Pr  cision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les r  sultats d  taill  s
print(f"Pr  cision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['l  ', 'vraies valeurs'])
plt.xlabel('Pr  dictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# R  cup  rer l'importance des features
# R  cup  rer le mod  le du classifieur apr  s l'entra  nement

```

```

model = pipeline.named_steps['classifieur']

# Accéder aux importances des features
importances = model.feature_importances_

# Récupérer les noms des features après transformation
# Utiliser l'encodeur OneHotEncoder pour gérer les variables catégorielles en
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la t

# Les noms des features après transformation
features = np.concatenate([cat_columns, cyclical_columns, passthrough_featu

# Tracer l'importance des features
plt.figure(figsize=(12, 12))
plt.barh(features, importances)
plt.xlabel("Importance des Features")
plt.title("Importance des features dans le modèle Random Forest avec oversam
plt.show()

```

```

Accuracy du modèle : 0.69
Précision : 0.66
Rappel : 0.69
F1-score : 0.67

Rapport de Classification :

```

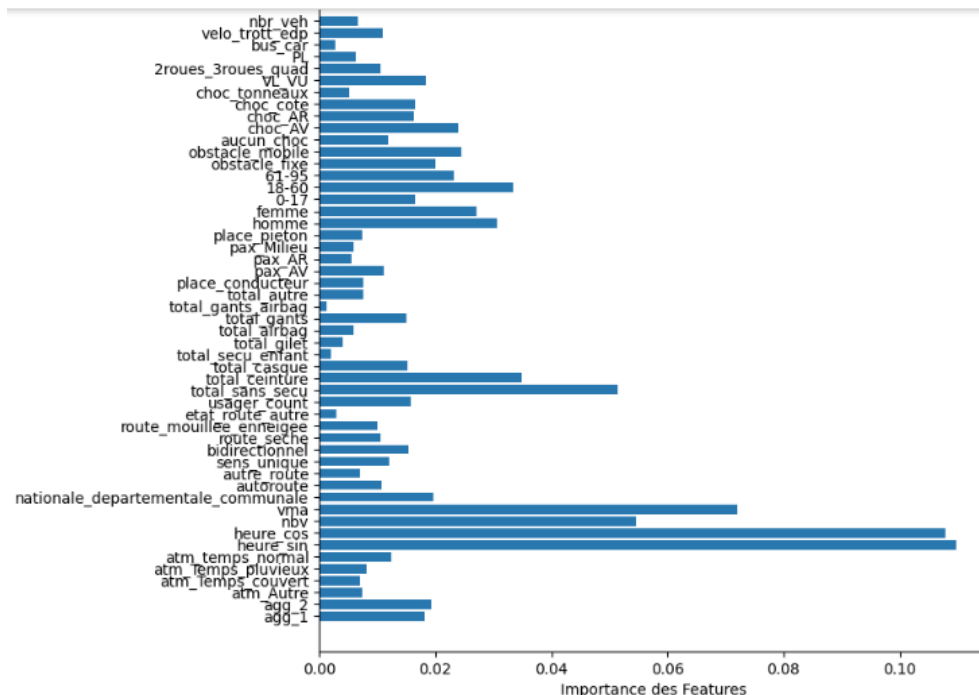
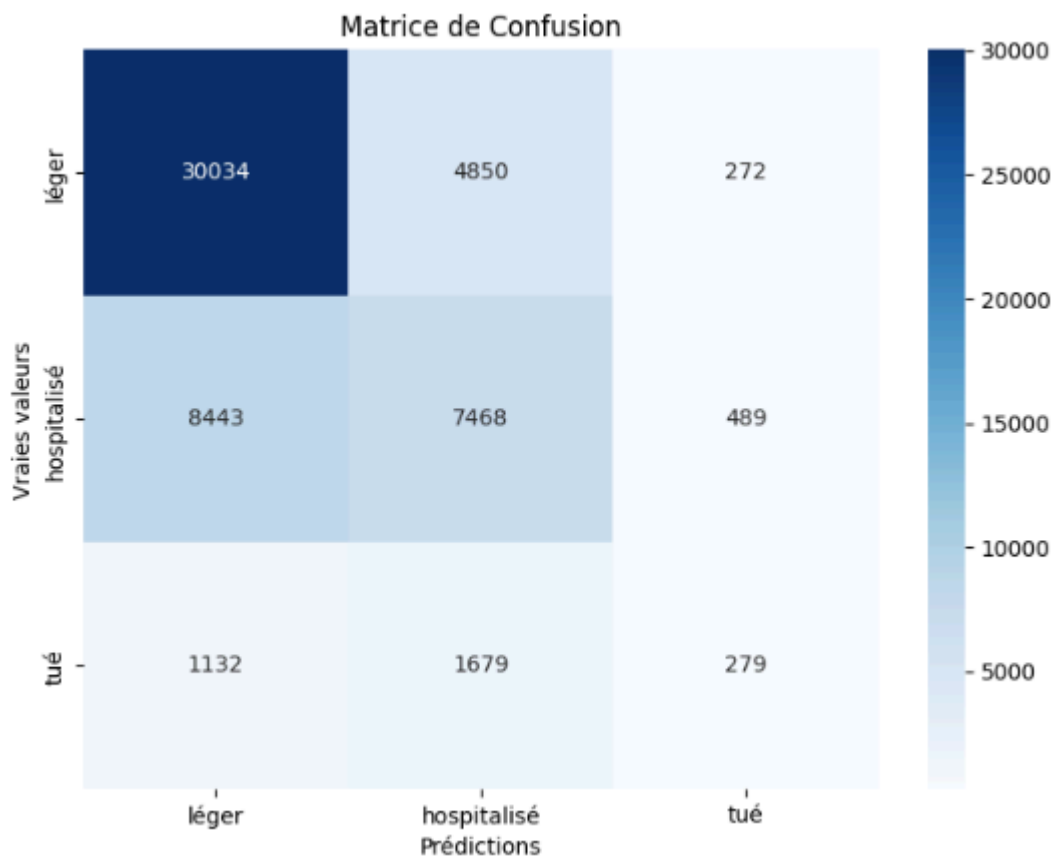
	precision	recall	f1-score	support
0	0.76	0.85	0.80	35156
1	0.53	0.46	0.49	16400
2	0.27	0.09	0.14	3090
accuracy			0.69	54646
macro avg	0.52	0.47	0.48	54646
weighted avg	0.66	0.69	0.67	54646

Recall classe 1: 0.46

F1\_Score classe 1: 0.49

Nous essaierons d'améliorer ces scores





On constate que les 5 variables les plus importantes sont : vma, nbv, heure, total\_sans\_secu, nationale\_communale\_departementale. Nous allons essayer un

modèle avec ces variables uniquement, pour voir si les résultats pour la classe 1 sont identiques.

## Random Forest avec les 5 variables les plus importantes

```
# Préparation des données
X = df.drop(['Num_Acc', 'total_ceinture', 'total_casque',
            'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
            'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
            'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
            '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
            'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
            '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh', 'autoroute',
            'sens_unique', 'bidirectionnel', 'route_seche', 'atm',
            'route_mouillee_enneigee', 'etat_route_autre', 'usager_count', 'lat', 'long', 'jour_semaine'])
y = df['gravite_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
```

```

categorical_features = ['agg'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'total_sans_secu'] # À laisser inchangées

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classif', RandomForestClassifier(n_estimators=100, random_state=42))]

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés

```

```

print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['l'é', 'l', 'l'])
plt.xlabel('Préddictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

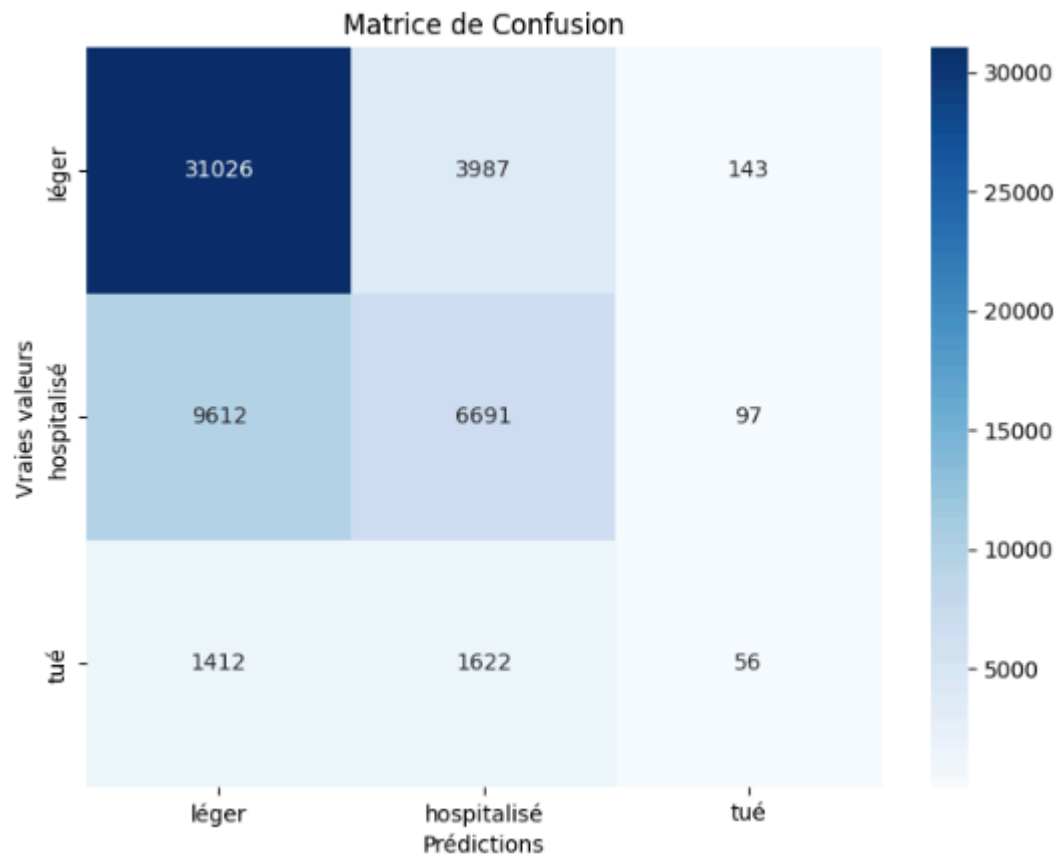
```

Accuracy du modèle : 0.69
Précision : 0.65
Rappel : 0.69
F1-score : 0.66

Rapport de Classification :

```

	precision	recall	f1-score	support
0	0.74	0.88	0.80	35156
1	0.54	0.41	0.47	16400
2	0.19	0.02	0.03	3090
accuracy			0.69	54646
macro avg	0.49	0.44	0.43	54646
weighted avg	0.65	0.69	0.66	54646



Recall classe 1: 0.41

F1\_Score classe 1: 0.47

Les scores sont plus faibles qu'avec toutes les variables. Nous conserverons donc toutes les variables pour la suite des essais.

## Random Forest avec Oversampling pour gérer le déséquilibre des classes

# Préparation des données

```
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int']
y = df['gravité_accident']
```

# Définition de la classe pour la transformation de l'heure

```
class CyclicalFeatures(BaseEstimator, TransformerMixin):
```

```
    """Transforme une colonne de type heure en variables cycliques sin et cos.
```

```
    def __init__(self, period=24):
```

```
        self.period = period
```

```

def fit(self, X, y=None):
    return self

def transform(self, X):
    X = X.astype(float) # Assurer que les valeurs sont numériques
    X_sin = np.sin(2 * np.pi * X / self.period)
    X_cos = np.cos(2 * np.pi * X / self.period)
    return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Appliquer SMOTE dans la pipeline (avant l'entraînement du modèle)
smote = SMOTE(random_state=42, sampling_strategy = 'auto', k_neighbors = 5)

# Définir la pipeline complète avec SMOTE
pipeline = imPipeline([

```

```

('preprocessor', preprocessor), # Transformation des variables
('smote', smote), # Oversampling via SMOTE
('classifieur', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Entraîner le modèle avec oversampling
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle avec oversampling : {accuracy:.2f}")

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")

```

```

print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['l'é',
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# Récupérer l'importance des features
# Récupérer le modèle du classifieur après l'entraînement
model = pipeline.named_steps['classifieur']

# Accéder aux importances des features
importances = model.feature_importances_

# Récupérer les noms des features après transformation
# Utiliser l'encodeur OneHotEncoder pour gérer les variables catégorielles en
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la t

# Les noms des features après transformation
features = np.concatenate([cat_columns, cyclical_columns, passthrough_featu

# Tracer l'importance des features
plt.figure(figsize=(12, 12))
plt.barh(features, importances)
plt.xlabel("Importance des Features")
plt.title("Importance des features dans le modèle Random Forest avec oversar
plt.show()

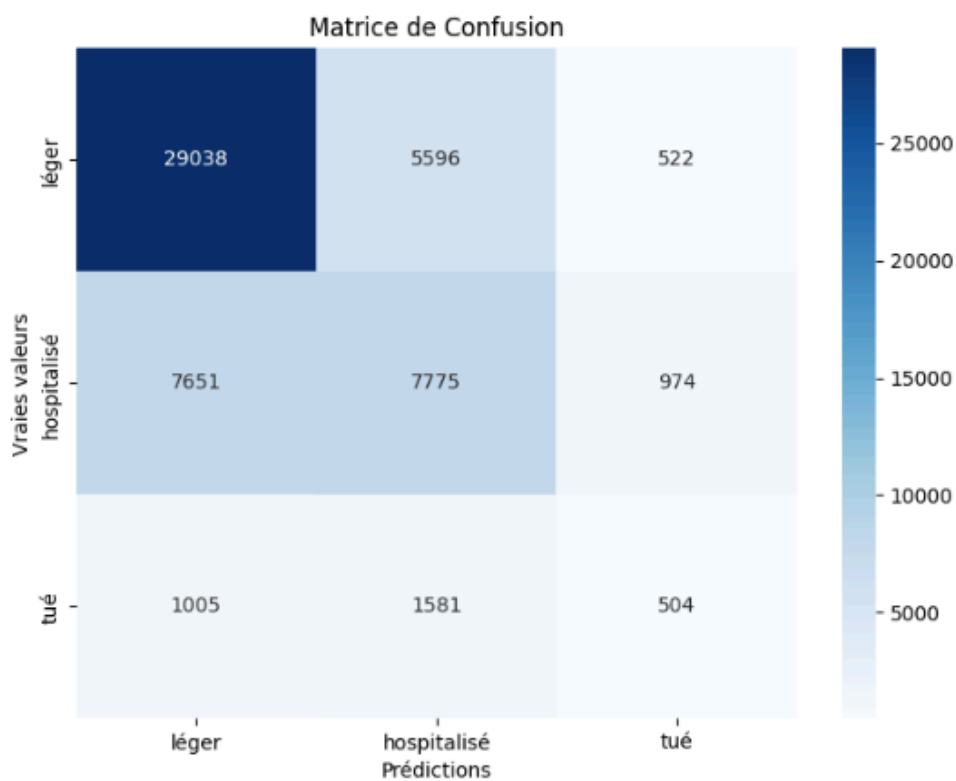
```



Accuracy du modèle avec oversampling : 0.68  
 Accuracy du modèle : 0.68  
 Précision : 0.67  
 Rappel : 0.68  
 F1-score : 0.67

Rapport de Classification :

	precision	recall	f1-score	support
0	0.77	0.83	0.80	35156
1	0.52	0.47	0.50	16400
2	0.25	0.16	0.20	3090
accuracy			0.68	54646
macro avg	0.51	0.49	0.50	54646
weighted avg	0.67	0.68	0.67	54646



Recall classe 1: 0.47

F1\_Score classe 1: 0.50

Les score sont un peu meilleurs avec l'oversampling

## Random Forest avec Undersampling

# Préparation des données

```

X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int']
y = df['gravité_accident']

```

```

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

```

```

# Appliquer RandomUnderSampler dans la pipeline (avant l'entraînement du modèle)
under_sampler = RandomUnderSampler(random_state=42)

# Définir la pipeline complète avec undersampling
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('under_sampler', under_sampler), # Undersampling via RandomUnderSampler
    ('classif', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Entraîner le modèle avec undersampling
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accurac   du mod  le avec undersampling : {accuracy:.2f}")

# Pr  cision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les r  sultats d  taill  s
print(f"Pr  cision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

```

```

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['l\'é
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# Récupérer l'importance des features
# Récupérer le modèle du classifieur après l'entraînement
model = pipeline.named_steps['classifieur']

# Accéder aux importances des features
importances = model.feature_importances_

# Récupérer les noms des features après transformation
# Utiliser l'encodeur OneHotEncoder pour gérer les variables catégorielles en
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la tr

# Les noms des features après transformation
features = np.concatenate([cat_columns, cyclical_columns, passthrough_featu

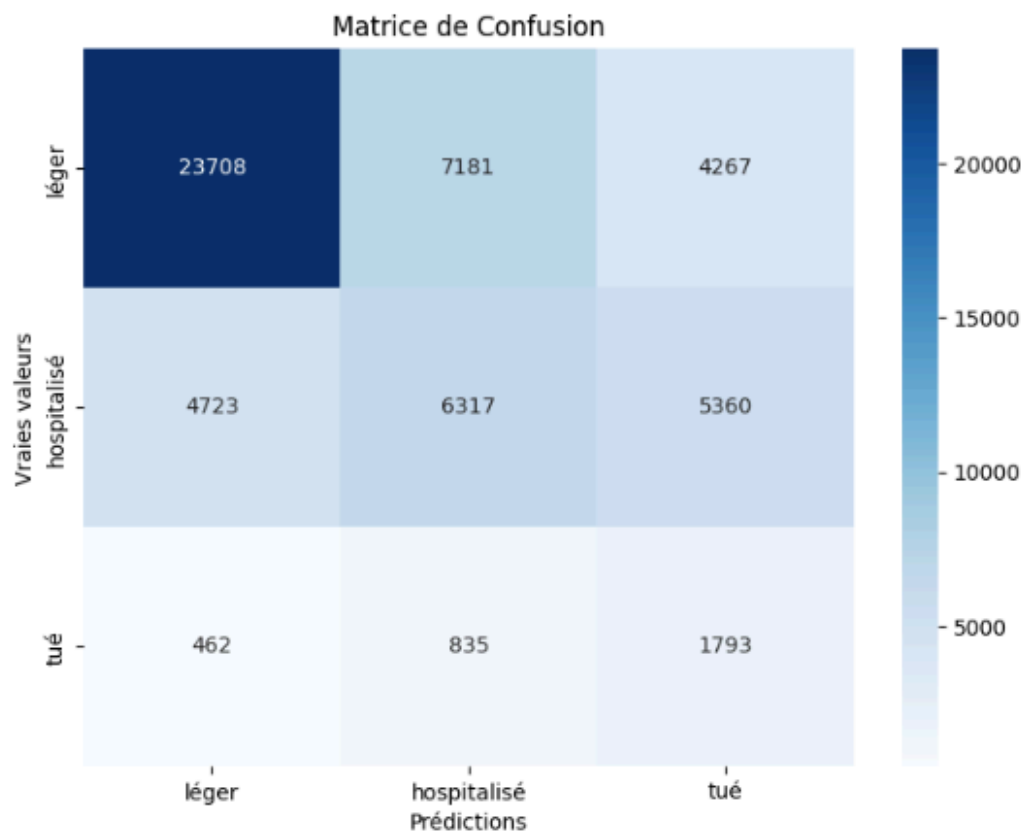
# Tracer l'importance des features
plt.figure(figsize=(12, 12))
plt.barh(features, importances)
plt.xlabel("Importance des Features")
plt.title("Importance des features dans le modèle Random Forest avec undersa
plt.show()

```

Accuracy du modèle avec undersampling : 0.58  
 Précision : 0.67  
 Rappel : 0.58  
 F1-score : 0.61

Rapport de Classification :

	precision	recall	f1-score	support
0	0.82	0.67	0.74	35156
1	0.44	0.39	0.41	16400
2	0.16	0.58	0.25	3090
accuracy			0.58	54646
macro avg	0.47	0.55	0.47	54646
weighted avg	0.67	0.58	0.61	54646



Recall classe 1: 0.39

F1\_Score classe 1: 0.41

Les scores sont bien inférieurs aux tests précédents.

**Grid search pour optimiser paramètres du modele avec toutes les variables. Utilisation de**

## class\_weight='balanced' pour gérer le déséquilibre des classes

```
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
```

```

'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
'2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

# 📌 **Pipeline de preprocessing**
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)
], remainder='passthrough')

# 📌 **Modèle de classification**
classifieur = RandomForestClassifier(random_state=42, class_weight='balanced')

# 📌 **Pipeline principal avec imblearn Pipeline**
pipeline = Pipeline([
    ('preprocessor', preprocessor), # Étape de transformation
    ('classifieur', classifieur) # Modèle final
])

# 📌 **Définition de la grille de recherche**
param_grid = {
    'classifieur__n_estimators': [50, 100, 200], # Nombre d'arbres
    'classifieur__max_depth': [10, 20, None], # Profondeur maximale
    'classifieur__class_weight': ['balanced', None] # Tester 'balanced' et None
}

# 📌 **Lancer la GridSearchCV**
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='f1_weighted')
grid_search.fit(X_train, y_train)

# 📌 **Afficher les meilleurs paramètres**
print("Meilleurs paramètres:", grid_search.best_params_)

# 📌 **Évaluation du modèle optimal**
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

```

```

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))

# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()

```

Meilleurs paramètres: {'classifier\_\_class\_weight': None, 'classifier\_\_max\_depth': 20, 'classifier\_\_n\_estimators': 100}

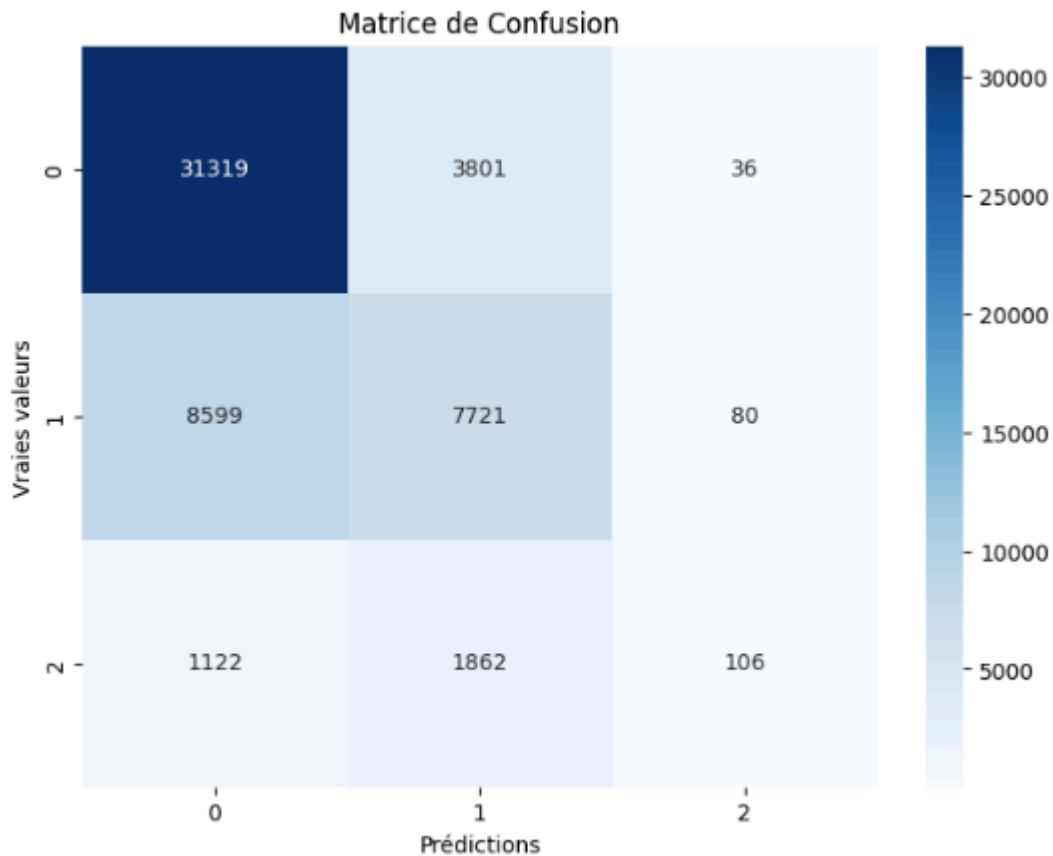
📊 \*\*Performance du modèle optimal\*\*

Accuracy: 0.72  
Precision: 0.69  
Recall: 0.72  
F1-score: 0.69

Rapport de Classification:

	precision	recall	f1-score	support
0	0.76	0.89	0.82	35156
1	0.58	0.47	0.52	16400
2	0.48	0.03	0.06	3090
accuracy			0.72	54646
macro avg	0.61	0.47	0.47	54646
weighted avg	0.69	0.72	0.69	54646





Recall classe 1: 0.47

F1\_Score classe 1: 0.52

Ces paramètres améliorent encore les scores

**Random Forest avec définition d'une métrique qui donne plus de poids à la classe 'blessé hospitalisé' pour le F1\_score, en utilisant les meilleurs paramètres retenus précédemment lors de la gridsearch.**

```
# Meilleurs paramètres: {'classifier__max_depth': 10, 'classifier__n_estimators':
```

```
# ON va créer une métrique personnalisée qui donne plus de poids à la prédic
```

```
from sklearn.model_selection import train_test_split, cross_val_score
```

```
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.compose import ColumnTransformer
```

```
from sklearn.pipeline import Pipeline
```

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

# Préparation des données
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int']
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',

```

```

'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
'18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
'2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Préparation des données
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
            'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_er

```

```
'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque',
'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'tota
'pax_AV', 'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femm
'obstacle_fixe', 'obstacle_mobile', 'aucun_choc', 'choc_AV', 'cho
'VL_VU', '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', '

# Appliquer le ColumnTransformer avec `remainder='passthrough`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)],
    remainder='passthrough') # Laisser les autres variables inchangées

# Définir les différentes pondérations à tester
class_weights_list = [
    {0: 1, 1: 2, 2: 1},
    {0: 1, 1: 10, 2: 5},
    {0: 1, 1: 10, 2: 10},
    {0: 1, 1: 10, 2: 5},
]

# Définir une fonction pour la précision pondérée
def weighted_f1_score(y_true, y_pred, class_weights):
    # Calculer la précision pour chaque classe
    f1_per_class = f1_score(y_true, y_pred, average=None)

    # Associer les poids aux classes dans le même ordre que dans precision_per_class
    weights = [class_weights.get(i, 1) for i in range(len(f1_per_class))]

    # Calculer la précision pondérée en fonction des poids
    weighted_f1 = np.dot(f1_per_class, weights) / sum(weights)
    return weighted_f1

# Définir la pipeline complète
pipeline = Pipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('classifier', RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42))
])
```

```

# Pour chaque configuration de poids, effectuer une validation croisée et calculer les
results = {}

for class_weights in class_weights_list:
    # Utiliser cross_val_score pour évaluer la performance avec les poids de classe
    f1_scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='f1_weighted',
                                class_weight=class_weights)

    # Enregistrer la moyenne des scores de précision pondérée
    results[str(class_weights)] = np.mean(f1_scores)

# Afficher les résultats pour chaque configuration de pondération
for class_weights, score in results.items():
    print(f"Configuration des poids {class_weights} → Précision pondérée : {score}")

# Trouver la configuration qui donne la meilleure précision pondérée
best_class_weights = max(results, key=results.get)
print(f"\nMeilleure configuration de pondération : {best_class_weights}")

# Créer la pipeline en utilisant les poids optimaux dans le classificateur Random Forest
pipeline_with_optimal_weights = Pipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('classifier', RandomForestClassifier(
        n_estimators=50,
        max_depth=10,
        random_state=42,
        class_weight=eval(best_class_weights) # Appliquer les poids ici
    ))
])

# Entraîner le modèle avec les poids optimaux
pipeline_with_optimal_weights.fit(X_train, y_train)

# Faire des prédictions sur l'ensemble de test
y_pred = pipeline_with_optimal_weights.predict(X_test)

# Evaluation de la performance du modèle

```

```

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1_optimal = f1_score(y_test, y_pred, average='weighted')

print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score pondéré avec les poids optimaux: {f1_optimal}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))

# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()

```

```

Configuration des poids {0: 1, 1: 2, 2: 1} -> Précision pondérée : 0.44670245271593345
Configuration des poids {0: 1, 1: 10, 2: 5} -> Précision pondérée : 0.3539732055397752
Configuration des poids {0: 1, 1: 10, 2: 10} -> Précision pondérée : 0.27334463537080333

```

```

Meilleure configuration de pondération : {0: 1, 1: 2, 2: 1}

```

```

📊 **Performance du modèle optimal**
Accuracy: 0.70
Precision: 0.70
Recall: 0.70
F1-score pondéré avec les poids optimaux: 0.6825473378851216

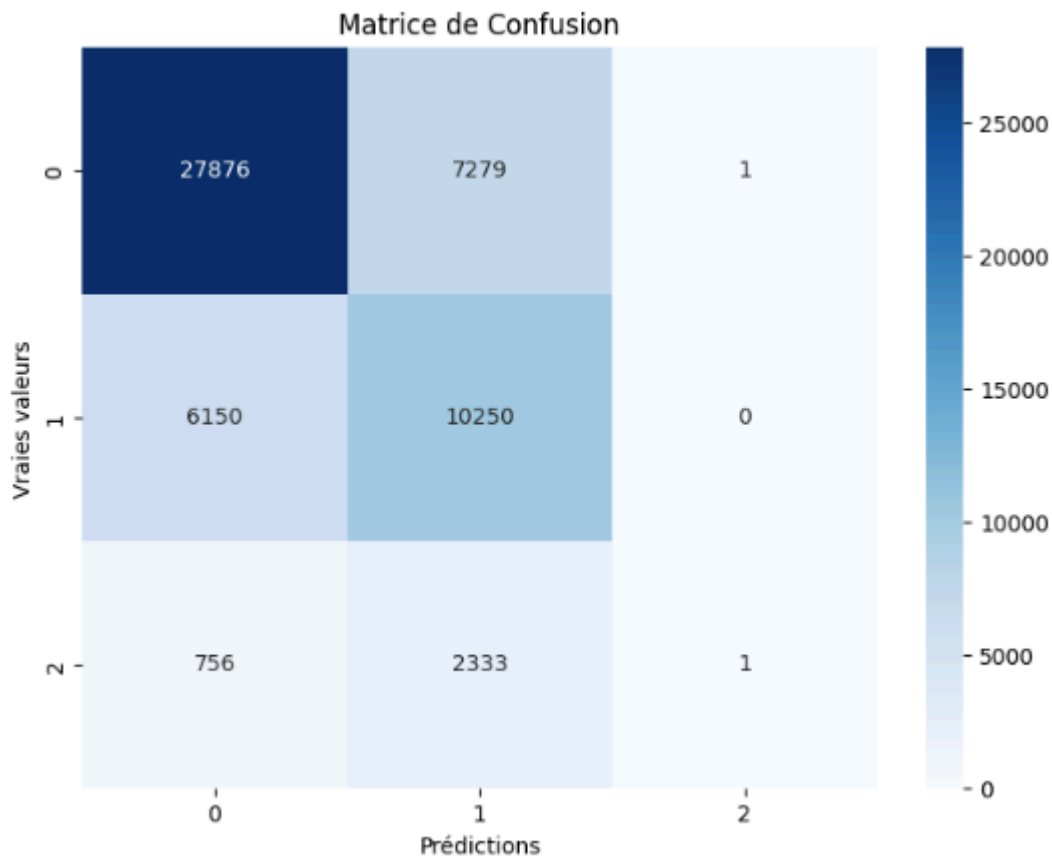
```

```

Rapport de Classification:

```

	precision	recall	f1-score	support
0	0.80	0.79	0.80	35156
1	0.52	0.62	0.57	16400
2	0.50	0.00	0.00	3090
accuracy			0.70	54646
macro avg	0.61	0.47	0.45	54646
weighted avg	0.70	0.70	0.68	54646



Recall classe 1: 0.62

F1\_Score classe 1: 0.57

Les scores sont nettement améliorés

**Random Forest avec définition d'une métrique qui donne plus de poids à la classe 'blessé hospitalisé' pour le Recall, en utilisant les meilleurs paramètres retenus précédemment lors de la gridsearch.**

```
from sklearn.metrics import recall_score
```

```
# Définir une fonction pour le rappel pondéré
```

```
def weighted_recall(y_true, y_pred, class_weights):
```

```

# Calculer le rappel pour chaque classe
recall_per_class = recall_score(y_true, y_pred, average=None)

# Associer les poids aux classes dans le même ordre que dans recall_per_c
weights = [class_weights.get(i, 1) for i in range(len(recall_per_class))]

# Calculer le rappel pondéré en fonction des poids
weighted_recall_score = np.dot(recall_per_class, weights) / sum(weights)
return weighted_recall_score

# Définir les différentes pondérations à tester (déjà définies dans ton code pré
class_weights_list = [
    {0: 1, 1: 2, 2: 1},
    {0: 1, 1: 10, 2: 5},
    {0: 1, 1: 10, 2: 10},
    {0: 1, 1: 10, 2: 5},
]

# Initialiser un dictionnaire pour stocker les résultats du rappel pondéré
results_recall = {}

# Pour chaque configuration de poids, effectuer une validation croisée et calc
for class_weights in class_weights_list:
    # Utiliser cross_val_score pour évaluer la performance avec les poids de cla
    recall_scores = cross_val_score(
        pipeline, X_train, y_train, cv=5,
        scoring=lambda est, X, y: weighted_recall(y, est.predict(X), class_weights
    )

    # Enregistrer la moyenne des scores de rappel pondéré
    results_recall[str(class_weights)] = np.mean(recall_scores)

# Afficher les résultats pour chaque configuration de pondération
for class_weights, score in results_recall.items():
    print(f"Configuration des poids {class_weights} → Rappel pondéré : {score}")

# Trouver la configuration qui donne le meilleur rappel pondéré

```



```

best_class_weights_recall = max(results_recall, key=results_recall.get)
print(f"\nMeilleure configuration de pondération (Rappel) : {best_class_weight}

# Créer la pipeline en utilisant les poids optimaux dans le classificateur Rando
pipeline_with_optimal_weights = Pipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('classifient', RandomForestClassifier(
        n_estimators=50,
        max_depth=10,
        random_state=42,
        class_weight=eval(best_class_weights) # Appliquer les poids ici
    ))
])

# Entraîner le modèle avec les poids optimaux
pipeline_with_optimal_weights.fit(X_train, y_train)

# Faire des prédictions sur l'ensemble de test
y_pred = pipeline_with_optimal_weights.predict(X_test)

# Evaluation de la performance du modèle

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall_optimal = recall_score(y_test, y_pred, average='weighted')
f1_score = f1_score(y_test, y_pred, average='weighted')

print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall_optimal: {recall:.2f}")
print(f"F1-score: {f1_score:.2f}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))


# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))

```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()
```

```
Configuration des poids {0: 1, 1: 2, 2: 1} -> Rappel pondéré : 0.43160848418020964
Configuration des poids {0: 1, 1: 10, 2: 5} -> Rappel pondéré : 0.3113245375793031
Configuration des poids {0: 1, 1: 10, 2: 10} -> Rappel pondéré : 0.23904561744646738
```

```
Meilleure configuration de pondération (Rappel) : {0: 1, 1: 2, 2: 1}
```

```
 **Performance du modèle optimal**
```

```
Accuracy: 0.70
```

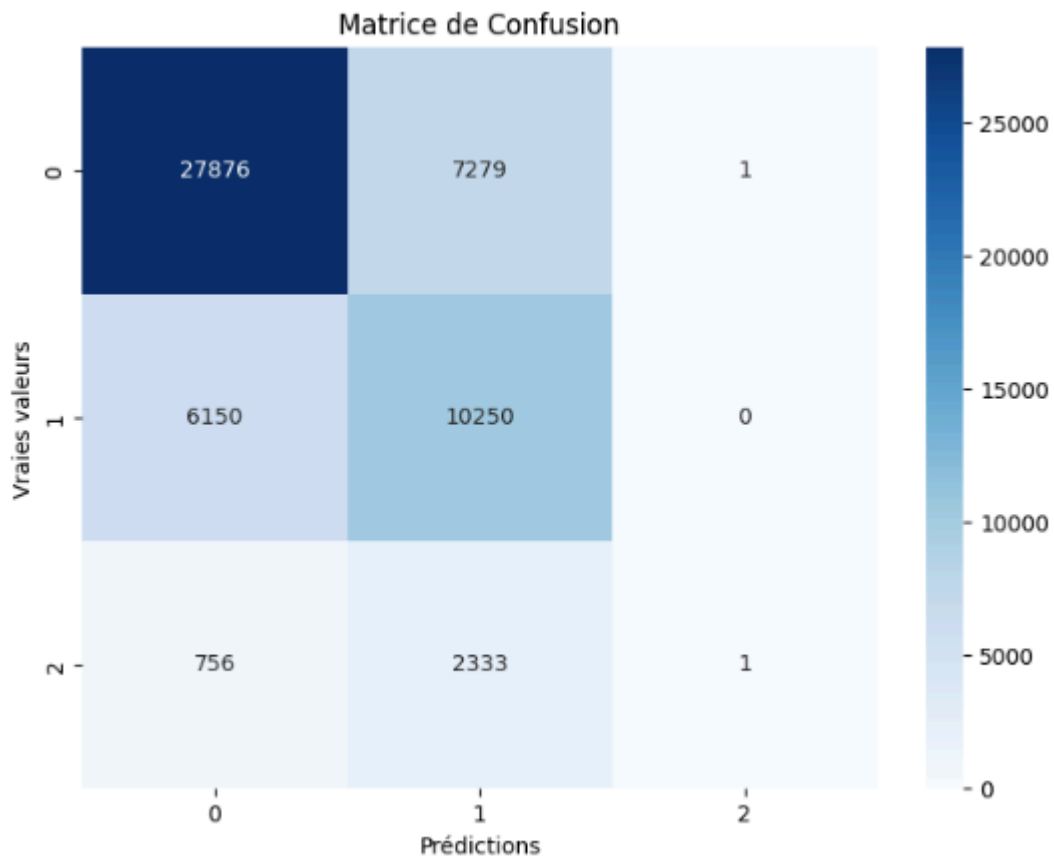
```
Precision: 0.70
```

```
Recall_optimal: 0.70
```

```
F1-score: 0.68
```

```
Rapport de Classification:
```

	precision	recall	f1-score	support
0	0.80	0.79	0.80	35156
1	0.52	0.62	0.57	16400
2	0.50	0.00	0.00	3090
accuracy			0.70	54646
macro avg	0.61	0.47	0.45	54646
weighted avg	0.70	0.70	0.68	54646



Recall classe 1: 0.62

F1\_Score classe 1: 0.57

Scores identiques au test précédent

### Conclusion du modèle Random Forest:

Les meilleurs scores obtenus pour le modèle Random Forest sont ceux avec une métrique pondérée pour optimiser le recall (ou le f1\_score), et les meilleurs paramètres obtenus avec une grille search. Les résultats optimaux sont un recall de 0.62 et un F1\_Score de 0.57, pour les paramètres suivants:

`{'classifier__max_depth': 10, 'classifier__n_estimators': 50}`

Pondération des classes pour optimiser la métrique: `{0: 1, 1: 2, 2: 1}`

## IV. Modèle 3 classes Régression Logistique

```
# Importation du df_machine learning
from google.colab import drive
import pandas as pd
import os
from io import StringIO
# Monter Google Drive
drive.mount('/content/drive', force_remount= True) #force_remount = True per
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Data:
df['gravité_accident'] = df['gravité_accident']-2
df.gravité_accident.value_counts()

# 0: blessé_léger
# 1: blessé_hospitalisé
# 2: tué
```

```
Mounted at /content/drive
count
gravité_accident
0      175525
1      82229
2      15472
dtype: int64
```

```
# Importation des bibliothèques nécessaires

import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```

from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline as imPipeline
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
import seaborn as sns
from sklearn.model_selection import GridSearchCV

```

## Régression logistique

```

# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int'])
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage

```

```

cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
    'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
    '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec la régression logistique
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classif', LogisticRegression(max_iter=1000, random_state=42))]) # Rég

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

```

```

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['bl', 'br', 'br', 'br'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

```

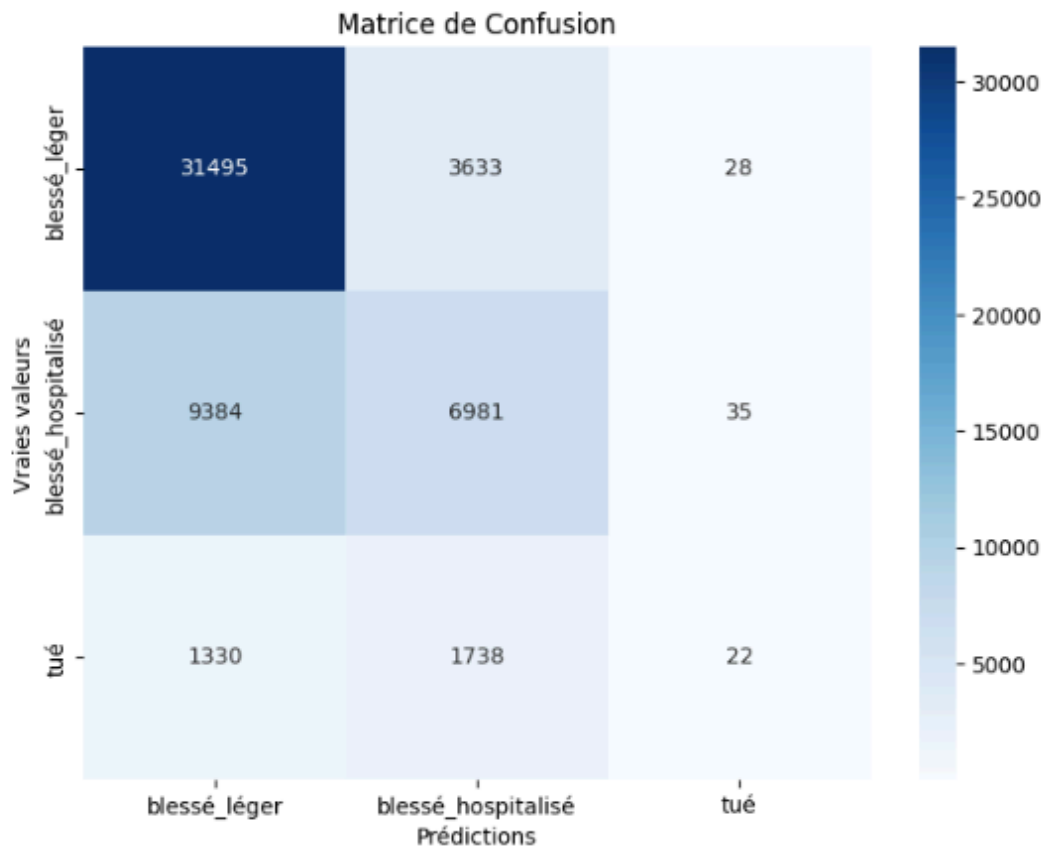
Accuracy du modèle : 0.70
Précision : 0.66
Rappel : 0.70
F1-score : 0.67

Rapport de Classification :
              precision    recall  f1-score   support

0             0.75         0.90         0.81     35156
1             0.57         0.43         0.49     16400
2             0.26         0.01         0.01       3090

 accuracy                   0.70     54646
 macro avg              0.52     0.44     0.44     54646
 weighted avg           0.66     0.70     0.67     54646

```



Recall classe 1: 0.43

F1\_Score classe 1: 0.49

## Logistic Régression avec oversampling

```
# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc','lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
y = df['gravité_accident']
```

```
# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self
```



```

def transform(self, X):
    X = X.astype(float) # Assurer que les valeurs sont numériques
    X_sin = np.sin(2 * np.pi * X / self.period)
    X_cos = np.cos(2 * np.pi * X / self.period)
    return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec SMOTE (oversampling) et la régression log
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('smote', SMOTE(random_state=42)), # Oversampling avec SMOTE
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression
])

# Entraîner le modèle
pipeline.fit(X_train, y_train)

```

```

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accurancy du modèle avec oversampling : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

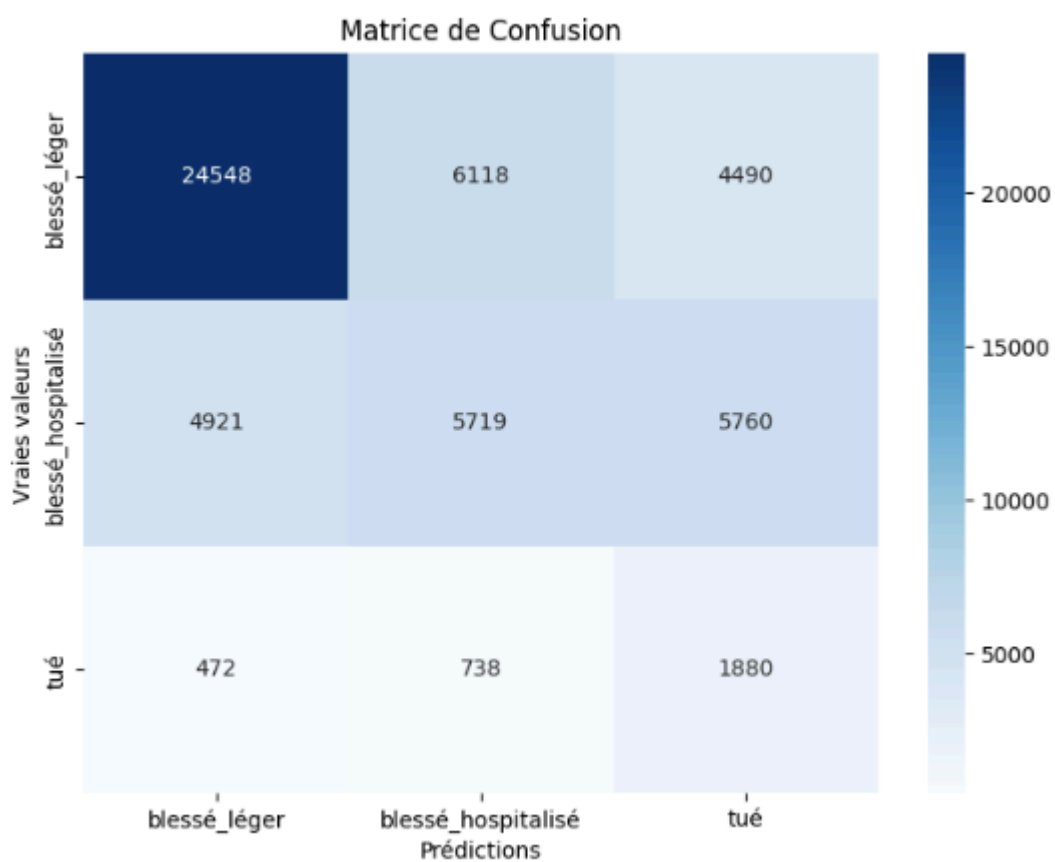
# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['bl', 'br', 'br', 'br'])
plt.xlabel('Prédiction')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

Accuracy du modèle avec oversampling : 0.59  
 Précision : 0.67  
 Rappel : 0.59  
 F1-score : 0.62

Rapport de Classification :

	precision	recall	f1-score	support
0	0.82	0.70	0.75	35156
1	0.45	0.35	0.39	16400
2	0.15	0.61	0.25	3090
accuracy			0.59	54646
macro avg	0.48	0.55	0.47	54646
weighted avg	0.67	0.59	0.62	54646



Recall classe 1: 0.35

F1\_Score classe 1: 0.39

L'oversampling dégrade les résultats

## Logistic Régression avec UnderSampling

```

# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc','lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma','nationale_departementale_communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
    'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
    '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laiss

# Appliquer le ColumnTransformer avec `remainder='passthrough'`

```

```

preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec RandomUnderSampler (undersampling) et
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression
])

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle avec undersampling : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

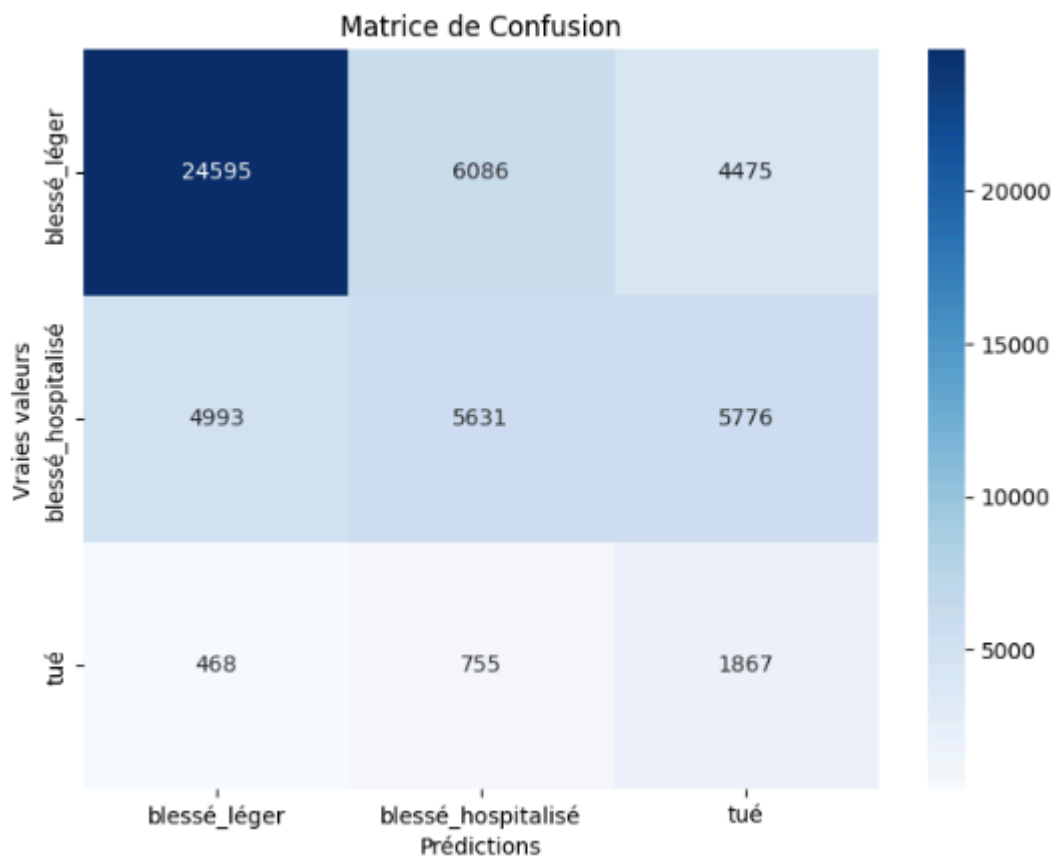
```

```
# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['bl
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()
```

```
Accuracy du modèle avec undersampling : 0.59
Précision : 0.67
Rappel : 0.59
F1-score : 0.62

Rapport de Classification :
```

	precision	recall	f1-score	support
0	0.82	0.70	0.75	35156
1	0.45	0.34	0.39	16400
2	0.15	0.60	0.25	3090
accuracy			0.59	54646
macro avg	0.47	0.55	0.46	54646
weighted avg	0.67	0.59	0.62	54646



Recall classe 1: 0.34

F1\_Score classe 1: 0.39

De même, l'undersampling dégrade les scores.

## Gridsearch pour optimiser les paramètres du modèle, avec `class_weight = 'balanced'` pour gérer le déséquilibre des classes

```
# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc','lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma','nationale_departementale_communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
```

```

'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
'18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
'2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec la régression logistique
pipeline = Pipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('classif', LogisticRegression(max_iter=1000, random_state=42, class_we
)])

# Paramètres à tester dans GridSearch
param_grid = {
    'classif__C': [0.01, 0.1, 1, 10, 100], # Paramètre C de la régression logistiqu
    'classif__solver': ['liblinear', 'saga'], # Différents solveurs pour la régress
}

# Définir GridSearchCV pour tester les différentes configurations
grid_search = GridSearchCV(
    pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)

# Entraîner le modèle avec GridSearch
grid_search.fit(X_train, y_train)

# Afficher les meilleurs paramètres trouvés
print("Meilleurs paramètres :")
print(grid_search.best_params_)

# Prédiction
y_pred = grid_search.predict(X_test)

```



```

# 📌 **Évaluation du modèle optimal**
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))

# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()

```

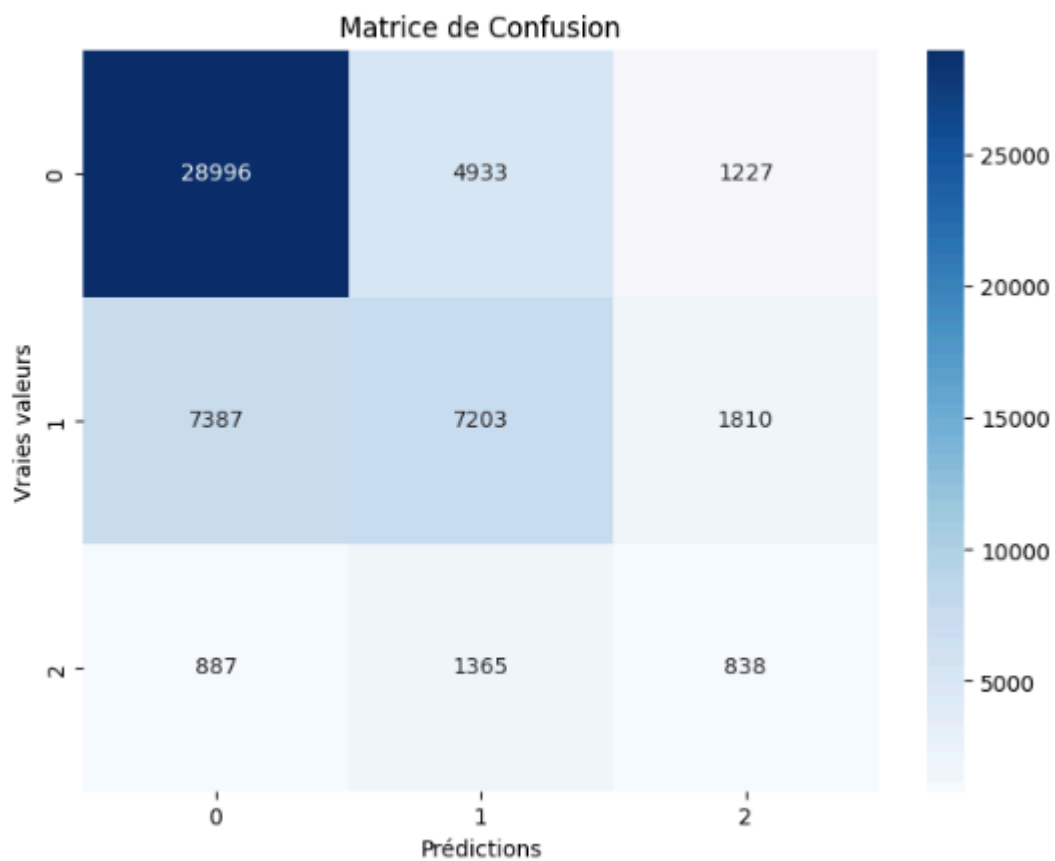
```
Meilleurs paramètres :
{'classifier__C': 0.01, 'classifier__solver': 'liblinear'}
```

```
 **Performance du modèle optimal**
Accuracy: 0.68
Precision: 0.67
Recall: 0.68
F1-score: 0.67
```

```
Rapport de Classification:
              precision    recall  f1-score   support

     0         0.78        0.82        0.80     35156
     1         0.53        0.44        0.48     16400
     2         0.22        0.27        0.24      3090

 accuracy          0.68        0.68        0.68     54646
  macro avg         0.51        0.51        0.51     54646
 weighted avg         0.67        0.68        0.67     54646
```



Recall classe 1: 0.44  
F1\_Score classe 1: 0.48

Impossible de faire d'autres tests avec métriques personnalisées, le noyau plante.

**Conclusion du modèle Régression Logistique:**  
Les meilleurs scores obtenus pour le modèle LogisticRegression sont ceux avec les paramètres suivants: {'classifier\_\_C': 0.01, 'classifier\_\_solver': 'liblinear'}.  
Les scores obtenus pour la classe blessés hospitalisés sont les suivants: Recall 0.44, et F1\_score 0.48.  
Ils restent nettement inférieurs aux meilleurs scores obtenus avec Random Forest

## V. Modèle 3 classes KNN

### Modèle KNN avec Gridsearch

```
# Importation du df_machine_learning

import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import matplotlib.pyplot as plt
import seaborn as sns

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float)
```

```

X_sin = np.sin(2 * np.pi * X / self.period)
X_cos = np.cos(2 * np.pi * X / self.period)
return np.c_[X_sin, X_cos]

# Chargement des données (Assurez-vous que df est défini)
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int'
y = df['gravité_accident']

# Séparation des données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Définition des caractéristiques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_enneigee', 'e
    'total_sans_secu', 'total_ceinture', 'total_casque', 'total_secu_enfant', 'tota
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV', 'pax_AR', '
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc', 'choc_AV
    '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh']

# Transformation des données
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features),
    ('num', StandardScaler(), passthrough_features)
])

# Définition du modèle KNN avec GridSearchCV
knn = KNeighborsClassifier()
param_grid = {'classifier__n_neighbors': [3, 5, 7, 9], 'classifier__weights': ['unif

# Pipeline avec le classificateur KNN
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', knn)
])

```

```

# GridSearch pour trouver les meilleurs hyperparamètres
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy',
                             return_best=True)
grid_search.fit(X_train, y_train)

# Meilleur modèle trouvé
best_model = grid_search.best_estimator_
print(f"Meilleurs paramètres : {grid_search.best_params_}")

# Prédiction
y_pred = best_model.predict(X_test)

# Évaluation du modèle
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy : {accuracy:.2f}")
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

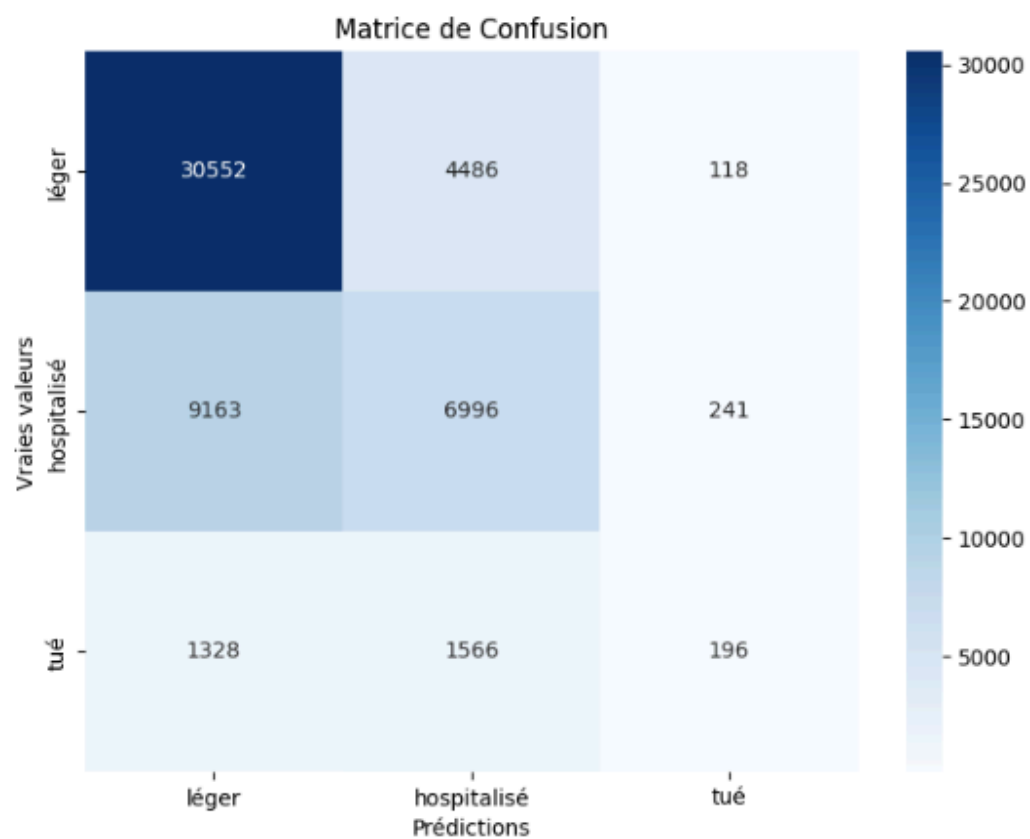
# Affichage de la matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['légende', 'non légende'], yticklabels=['légende', 'non légende'])
plt.xlabel('Prédiction')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

Meilleurs paramètres : {'classifier\_\_n\_neighbors': 9, 'classifier\_\_weights': 'uniform'}  
 Accuracy : 0.69  
 Précision : 0.66  
 Rappel : 0.69  
 F1-score : 0.66

Rapport de Classification :

	precision	recall	f1-score	support
0	0.74	0.87	0.80	35156
1	0.54	0.43	0.48	16400
2	0.35	0.06	0.11	3090
accuracy			0.69	54646
macro avg	0.54	0.45	0.46	54646
weighted avg	0.66	0.69	0.66	54646



Recall classe 1: 0.43  
 F1\_Score classe 1: 0.48

**KNN avec une métrique make\_scorer pour améliorer le F1 score et Gridsearch**

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import f1_score, make_scorer, confusion_matrix, classification_report
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.neighbors import KNeighborsClassifier

# =====
# 1 Transformation des Données
# =====

class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float)
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Chargement des données (remplace par ton DataFrame réel)
df = pd.read_csv("ton_fichier.csv") # Remplace par le vrai fichier

# Séparation des features et du target
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
            'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

# Décalage des labels (classes 2,3,4 deviennent 0,1,2)

```

```

y = y - 2

# Séparation en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Variables catégorielles et cycliques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_en',
                        'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'total',
                        'place_conducteur', 'pax_AV', 'pax_AR', 'pax_Milieu', 'place_piet',
                        '0-17', '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun',
                        'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU', '2roues_3roues',
                        'velo_trott_edp', 'nbr_veh']

# Transformation des données
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features),
    ('scaler', StandardScaler(), passthrough_features) # Normalisation pour KNN
])

X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# =====
# 2 Application de SMOTE pour équilibrer les classes
# =====
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)

# =====
# 3 Définition de la Métrique Personnalisée (F1-score pour une classe cible)
# =====
def f1_class_specific(y_true, y_pred, target_class=1):
    """

```



```

Fonction pour calculer le F1-score d'une classe spécifique.
"""
y_pred = (y_pred == target_class).astype(int)
y_true = (y_true == target_class).astype(int)
return f1_score(y_true, y_pred)

# Scorer pour GridSearchCV
f1_scorer = make_scorer(f1_class_specific, greater_is_better=True, target_clas

# =====
# 4 Entraînement du Modèle KNN avec GridSearchCV
# =====
knn_model = KNeighborsClassifier()

param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

grid_search = GridSearchCV(
    knn_model, param_grid, scoring=f1_scorer,
    cv=3, n_jobs=-1, verbose=1
)

grid_search.fit(X_train_balanced, y_train_balanced)

# Meilleurs hyperparamètres
best_params = grid_search.best_params_
print("Meilleurs hyperparamètres :", best_params)

# =====
# 5 Évaluation sur l'Ensemble de Test
# =====
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_transformed)

# Calcul du F1-score pour la classe cible

```

```

f1_test = f1_score(y_test == 1, y_pred == 1)
print(f"F1-score de la classe cible (accidents graves) sur test : {f1_test:.4f}")

# =====
# 6 Affichage de la Matrice de Confusion
# =====
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=[0
plt.xlabel("Prédictions")
plt.ylabel("Vraies Classes")
plt.title("Matrice de Confusion - KNN")
plt.show()

# Rapport de classification complet
print(classification_report(y_test, y_pred, target_names=["léger", "hospitalisé",

```

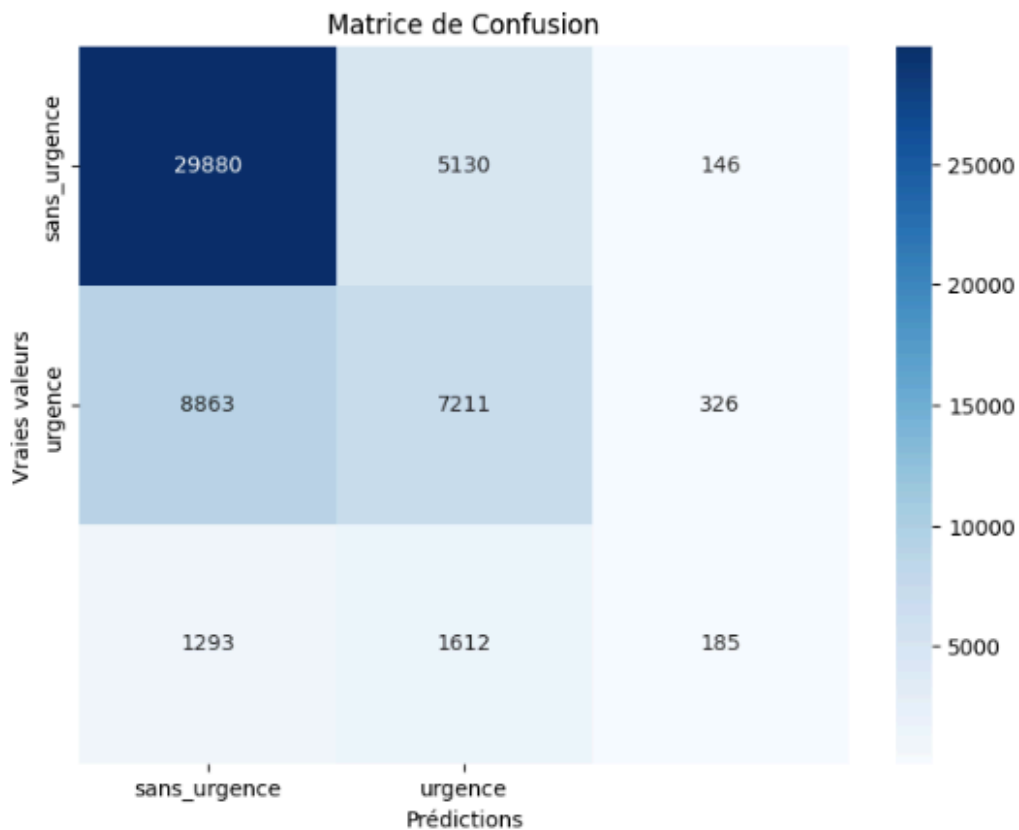
```

F1-score moyen pour la classe 1 : 0.4686
F1-score de la classe 1 sur l'ensemble de test : 0.4751

Rapport de Classification :

```

	precision	recall	f1-score	support
0	0.75	0.85	0.79	35156
1	0.52	0.44	0.48	16400
2	0.28	0.06	0.10	3090
accuracy			0.68	54646
macro avg	0.51	0.45	0.46	54646
weighted avg	0.65	0.68	0.66	54646



Recall classe 1: 0.44

F1\_Score classe 1: 0.48

Le score n'est pas amélioré avec cette métrique

## Conclusion du modèle KNN

Le modèle KNN fournit des score nettement moins bons que le Random Forest.

## VI. Modèle 3 classes XGBoost

```
# Importation du df_machine_learning
from google.colab import drive
import pandas as pd
import os
from io import StringIO
# Monter Google Drive
drive.mount('/content/drive', force_remount= True) #force_remount = True pe
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Data:
```

```
df.info()
df.gravité_accident.value_counts()
```

```
# Importation des bibliothèques nécessaires
import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from imblearn.pipeline import Pipeline as imPipeline # Utilisation d'un pipeline
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from sklearn.model_selection import GridSearchCV
```

## XGBOOST avec oversampling

```
from imblearn.over_sampling import SMOTE
from sklearn.pipeline import Pipeline

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
```

```

X_cos = np.cos(2 * np.pi * X / self.period)
return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparation des variables d'entrée et de sortie
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
            'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

# Décalage des labels pour correspondre à l'attente d'XGBoost ([2,3,4] → [0,1])
y = y - 2

# Séparation en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Variables à encoder
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_en',
                        'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'total',
                        'place_conducteur', 'pax_AV', 'pax_AR', 'pax_Milieu', 'place_pieton',
                        '0-17', '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun',
                        'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU', '2roues_3roues',
                        'velo_trott_edp', 'nbr_veh'] # À laisser inchangées

# Appliquer le ColumnTransformer pour transformer les variables
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)
], remainder='passthrough')

# Transformation des données d'entraînement
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# Application de SMOTE pour équilibrer les classes
smote = SMOTE(sampling_strategy='auto', random_state=42)

```

```

X_train_resampled, y_train_resampled = smote.fit_resample(X_train_transformed, y_train)

# Création et entraînement du modèle XGBoost
model = XGBClassifier(
    objective='multi:softmax', # Classification multiclass
    num_class=3, # Nombre de classes
    eval_metric='mlogloss', # Log loss
    use_label_encoder=False, # Suppression d'un avertissement lié à XGBoost
    random_state=42
)
model.fit(X_train_resampled, y_train_resampled)

# Prédiction (on reconstruit les classes en [2,3,4])
y_pred = model.predict(X_test_transformed) + 2

# Évaluation du modèle
print(f"Accurac   du mod  le : {accuracy_score(y_test + 2, y_pred):.2f}")
print("\nRapport de classification :")
print(classification_report(y_test + 2, y_pred))

# Visualisation de la matrice de confusion
conf_matrix = confusion_matrix(y_test + 2, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['l  ger', 'hospitalis  ', 'tu  '],
            yticklabels=['l  ger', 'hospitalis  ', 'tu  '])
plt.xlabel('Pr  dictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

```

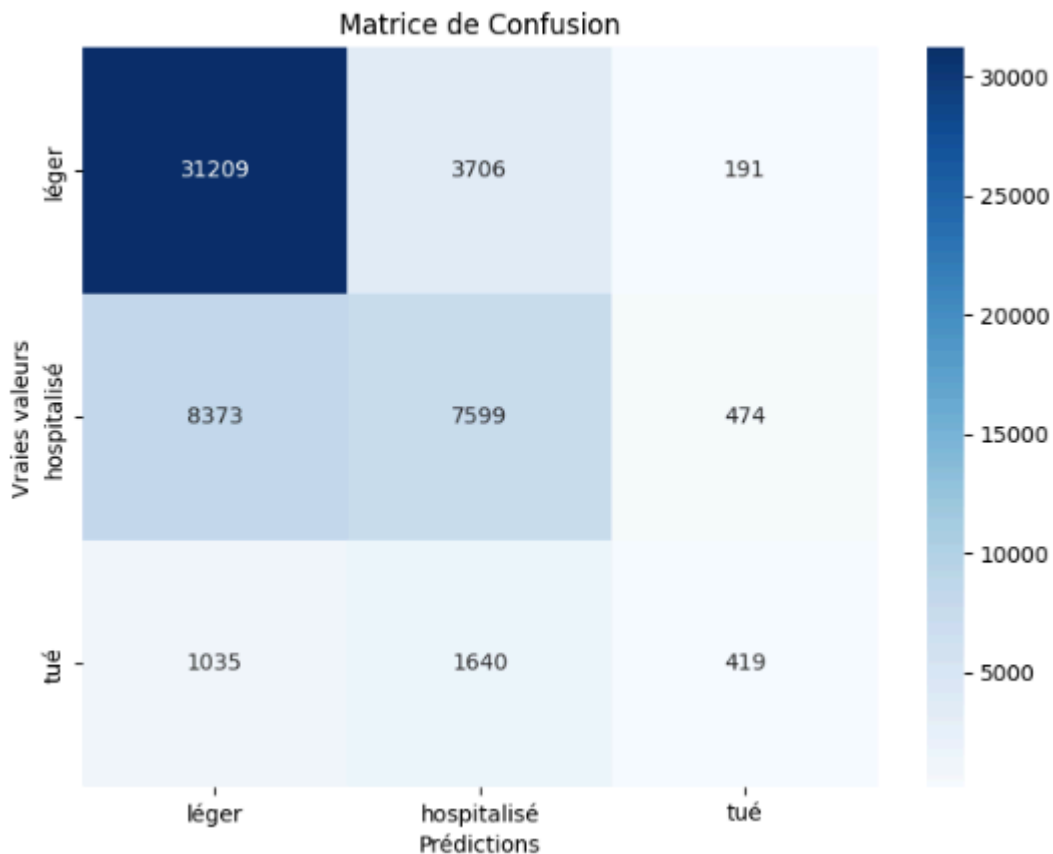
Accuracy du mod  le : 0.72

Rapport de classification :
              precision    recall  f1-score   support

     2         0.77         0.89         0.82       35106
     3         0.59         0.46         0.52       16446
     4         0.39         0.14         0.20        3094

 accuracy                   0.72       54646
 macro avg         0.58         0.50         0.51       54646
 weighted avg         0.69         0.72         0.70       54646

```



Recall classe 1: 0.46

F1\_Score classe 1: 0.52

## XGBOOST avec l'argument classweight pour gérer le déséquilibre des classes

```
# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
```

```

X_sin = np.sin(2 * np.pi * X / self.period)
X_cos = np.cos(2 * np.pi * X / self.period)
return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparation des variables d'entrée et de sortie
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
            'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

# Vérification des classes uniques
print("Classes uniques avant modification:", np.unique(y))

# Décalage des labels pour correspondre à l'attente d'XGBoost ([2,3,4] → [0,1])
y = y - 2

# Vérification après transformation
print("Classes uniques après transformation:", np.unique(y))

# Séparation en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Variables à encoder
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_en',
                        'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'total',
                        'place_conducteur', 'pax_AV', 'pax_AR', 'pax_Milieu', 'place_pieton',
                        '0-17', '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun',
                        'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU', '2roues_3roues',
                        'velo_trott_edp', 'nbr_veh'] # À laisser inchangées

# Appliquer le ColumnTransformer pour transformer les variables
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)
])

```



```

], remainder='passthrough')

# Calculer les poids des classes (pour gérer le déséquilibre)
class_weights = compute_sample_weight(class_weight='balanced', y=y_train)
print("Poids des classes calculés avec 'balanced' :", class_weights)

# Créer et entraîner le modèle XGBoost
model = XGBClassifier(
    objective='multi:softmax', # Classification multiclass
    num_class=3, # Nombre de classes
    eval_metric='mlogloss', # Log loss
    use_label_encoder=False, # Suppression d'un avertissement lié à XGBoost
    random_state=42
)

# Appliquer la transformation de prétraitement sur les données d'entraînement
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# Entraîner le modèle avec les poids de classes
model.fit(X_train_transformed, y_train, sample_weight=class_weights)

# Prédiction (on reconstruit les classes en [2,3,4])
y_pred = model.predict(X_test_transformed) + 2

# Évaluation du modèle
accuracy = accuracy_score(y_test + 2, y_pred) # On ajoute 2 à y_test pour correspondre
print(f"Accuracy du modèle : {accuracy:.2f}")

# Rapport de classification détaillé
print("\nRapport de classification :")
print(classification_report(y_test + 2, y_pred))

# Visualisation de la matrice de confusion
conf_matrix = confusion_matrix(y_test + 2, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['légende', 'prédiction'])

```

```

yticklabels=['léger', 'hospitalisé', 'tué'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

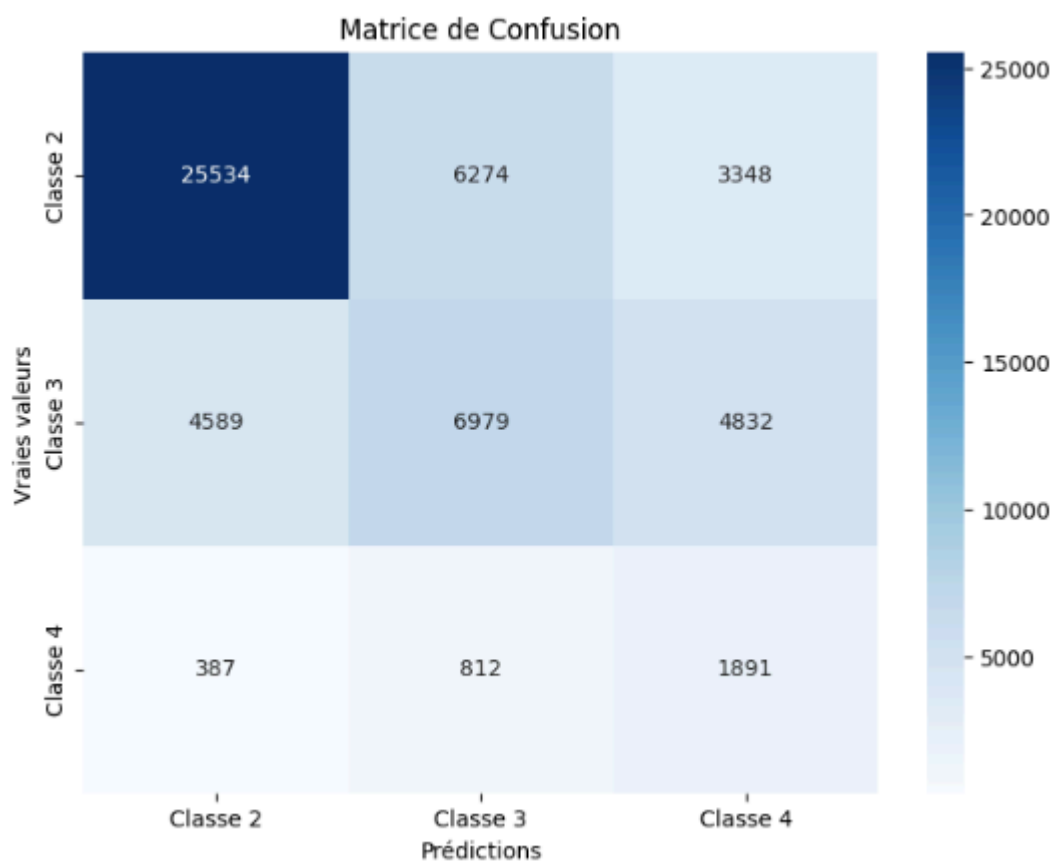
```

Accuracy du modèle : 0.63

Rapport de classification :

```

	precision	recall	f1-score	support
2	0.84	0.73	0.78	35156
3	0.50	0.43	0.46	16400
4	0.19	0.61	0.29	3090
accuracy			0.63	54646
macro avg	0.51	0.59	0.51	54646
weighted avg	0.70	0.63	0.65	54646



Recall classe 1: 0.43

F1\_Score classe 1: 0.46

Les scores sont moins bons qu'avec l'oversampling.

## Optimisation des hyperparamètres et classe weight

On pourrait ajouter beaucoup d'hyperparamètres mais ça plante systématiquement

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from xgboost import XGBClassifier
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin

# Définition de la transformation cyclique pour les variables horaires
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float)
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos]

# Séparation des variables d'entrée et de sortie
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
```

```

        'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

# Décalage des labels pour correspondre à l'attente d'XGBoost ([2,3,4] → [0,1])
y = y - 2

# Séparation en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et numériques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
numerical_features = ['nbv', 'vma', 'nationale_departementale_communale', 'a_sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_enr', 'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque', 'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'total_place_conducteur', 'pax_AV', 'pax_AR', 'pax_Milieu', 'place_pieton', '0-17', '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU', '2roues_3roues', 'velo_trott_edp', 'nbr_veh']

# Pipeline de transformation
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
        ('cyclical', CyclicalFeatures(), cyclical_features),
        ('num', StandardScaler(), numerical_features)
    ],
    remainder='passthrough' # Conserve les autres variables sans modification
)

# Construction du pipeline avec XGBoost
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifieur', XGBClassifier(objective='multi:softmax', num_class=3, eval_metric='logloss'))
])

# Définition de la grille d'hyperparamètres pour GridSearchCV

```

```

param_grid = {

    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [3, 6, 10],
    'classifier__scale_pos_weight': [1, 2, 5] # Gestion du déséquilibre des class
}

# Recherche des meilleurs paramètres avec GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='accuracy', )

# Entraîner le modèle avec la recherche en grille
grid_search.fit(X_train, y_train)

# Meilleurs paramètres et précision
print("Meilleurs paramètres trouvés : ", grid_search.best_params_)
print("Meilleure précision sur l'ensemble d'entraînement : ", grid_search.best_

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

# Rapport de classification
print("\nRapport de classification :")
print(classification_report(y_test, y_pred))

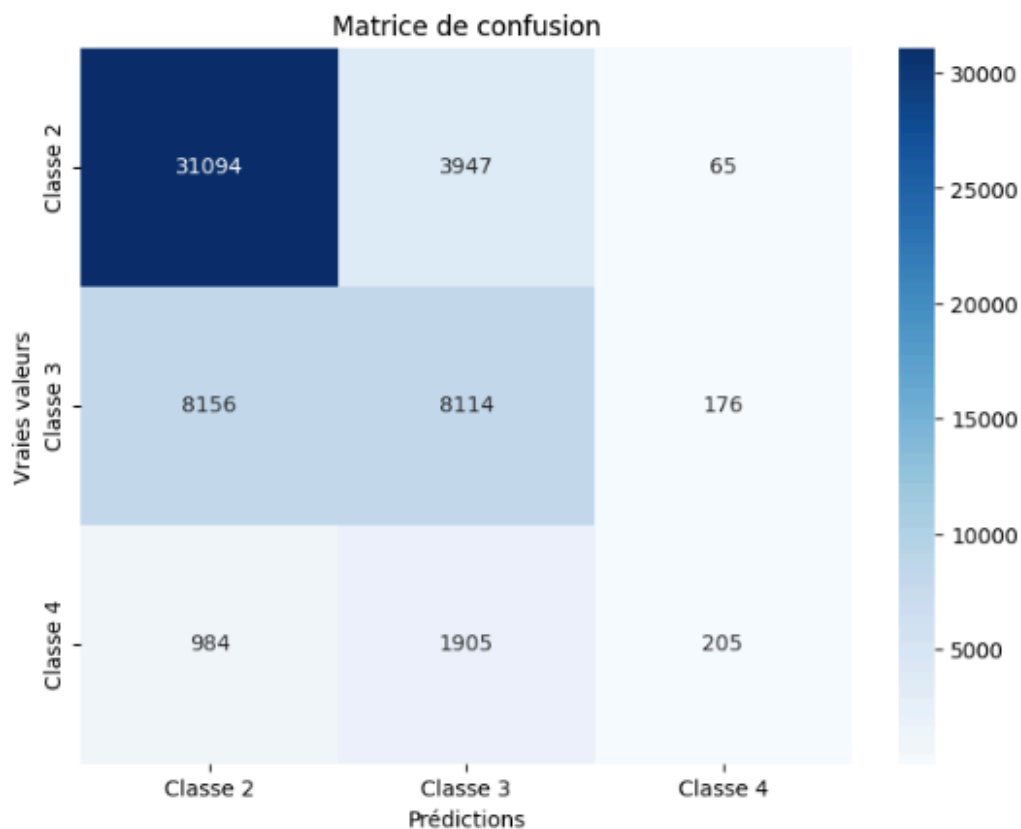
# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cl
plt.xlabel('Prédiction')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de confusion')
plt.show()

```

Meilleurs paramètres trouvés : {'classifier\_\_max\_depth': 6, 'classifier\_\_n\_estimators': 100, 'classifier\_\_scale\_pos\_weight': 1}  
 Meilleure précision sur l'ensemble d'entraînement : 0.7177006130478544

Rapport de classification :

	precision	recall	f1-score	support
0	0.77	0.89	0.83	35106
1	0.58	0.49	0.53	16446
2	0.46	0.07	0.12	3094
accuracy			0.72	54646
macro avg	0.60	0.48	0.49	54646
weighted avg	0.70	0.72	0.70	54646



Recall classe 1: 0.49

F1\_Score classe 1: 0.53

Les scores sont améliorés avec les paramètres optimaux

**XGBOOST Avec scoring personnalisé pour optimier le F1\_score de la classe 1**

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import f1_score, make_scorer, confusion_matrix, classification_report
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin
from xgboost import XGBClassifier

# =====
# 1 Transformation des Données
# =====

class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float)
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparation des features et du target
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
            'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

# Décalage des labels (classes 2,3,4 deviennent 0,1,2)
y = y - 2

```

```

# Séparation en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Variables catégorielles et cycliques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee',
                        'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'total_place_conducteur',
                        'pax_AV', 'pax_AR', 'pax_Milieu', 'place_pieton', '0-17', '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_obstacle',
                        'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU', '2roues_3roues', 'velo_trott_edp', 'nbr_veh']

# Transformation des données
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)
], remainder='passthrough')

X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# =====
# 2 Équilibrage des Classes avec SMOTE
# =====
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)

# =====
# 3 Définition de la Métrique Personnalisée (F1-score pour une classe cible)
# =====
def f1_xgb(preds, dtrain, target_class=1):
    labels = dtrain.get_label()
    preds = np.argmax(preds.reshape(len(labels), -1), axis=1)
    f1 = f1_score(labels == target_class, preds == target_class)

```



```

    return f'f1_class_{target_class}', f1

# Fonction pour GridSearchCV
def f1_class_specific(y_true, y_pred, target_class=1):
    y_pred = (y_pred == target_class).astype(int)
    y_true = (y_true == target_class).astype(int)
    return f1_score(y_true, y_pred)

f1_scorer = make_scorer(f1_class_specific, greater_is_better=True, target_class=1)

# =====
# 4 Entraînement du Modèle XGBoost avec GridSearchCV
# =====
xgb_model = XGBClassifier(
    objective="multi:softmax",
    num_class=3,
    eval_metric="mlogloss",
    use_label_encoder=False
)

param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200]
}

grid_search = GridSearchCV(
    xgb_model, param_grid, scoring=f1_scorer,
    cv=3, n_jobs=-1, verbose=1
)

grid_search.fit(X_train_balanced, y_train_balanced)

# Meilleurs hyperparamètres
best_params = grid_search.best_params_
print("Meilleurs hyperparamètres :", best_params)

# =====

```

```

# 5 Évaluation sur l'Ensemble de Test
# =====
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_transformed)

# Calcul du F1-score pour la classe cible
f1_test = f1_score(y_test == 1, y_pred == 1)
print(f"F1-score de la classe cible (accidents graves) sur test : {f1_test:.4f}")

# =====
# 6 Matrice de Confusion et Rapport de Classification
# =====
conf_matrix = confusion_matrix(y_test, y_pred)

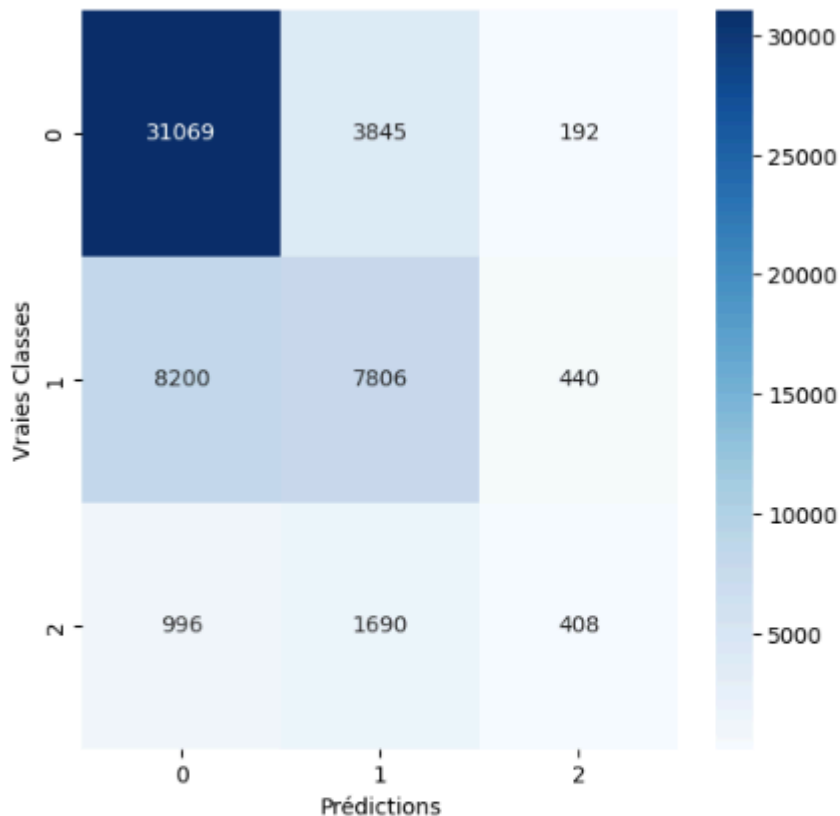
plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=[0,1], yticklabels=[0,1])
plt.xlabel("Prédictions")
plt.ylabel("Vraies Classes")
plt.title("Matrice de Confusion - XGBoost avec GridSearchCV")
plt.show()

# Rapport de classification
print("\nRapport de Classification :")
print(classification_report(y_test, y_pred, target_names=["léger", "hospitalisé"],

```

Meilleurs hyperparamètres : {'learning\_rate': 0.2, 'max\_depth': 7, 'n\_estimators': 200}  
 F1-score de la classe cible (accidents graves) sur test : 0.5241

Matrice de Confusion - XGBoost avec GridSearchCV



Rapport de Classification :

	precision	recall	f1-score	support
léger	0.77	0.89	0.82	35106
hospitalisé	0.59	0.47	0.52	16446
tué	0.39	0.13	0.20	3094
accuracy			0.72	54646
macro avg	0.58	0.50	0.52	54646
weighted avg	0.69	0.72	0.70	54646

Recall classe 1: 0.47

F1\_Score classe 1: 0.52

Le score est moins bon qu'avec class\_weight et gridsearch

## Conclusion du modèle XGBoost:

Les meilleurs scores obtenus pour le modèle XGBoost sont ceux avec les paramètres suivants: {'classifier\_\_max\_depth': 6, 'classifier\_\_n\_estimators': 100, 'classifier\_\_scale\_pos\_weight': 1}

**Les scores obtenus pour la classe blessés hospitalisés sont les suivants: Recall 0.47, et F1\_score 0.52. Ils restent inférieurs aux meilleurs scores obtenus avec Random Forest.**

Nous allons retester les différents modèles précédents, mais cette fois, notre variable cible va être séparée en deux classes: urgent (i.e. blessés hospitalisés) et non urgent (i.e. blessés légers ou tués)

## VII. Modèle 2 classes Random Forest

```
from google.colab import drive
import pandas as pd
import os
from io import StringIO
# Monter Google Drive
drive.mount('/content/drive', force_remount= True) #force_remount = True pe
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Data:
df['gravité_accident'] = df['gravité_accident'].replace({2: 'sans_urgence', 3: 'u
df['gravité_accident'].value_counts()
```

```
Mounted at /content/drive
count
gravité_accident
sans_urgence    190997
urgence         82229
dtype: int64
```

```
# Importation des bibliothèques nécessaires
import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

```

from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as imPipeline
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
import seaborn as sns
from sklearn.model_selection import GridSearchCV

```

## Random Forest simple

```

X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int'])
y = df['gravité_accident']

```

```

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
    'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
    '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classif', RandomForestClassifier(n_estimators=100, random_state=42))]

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

```

```

f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)


# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)


# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cl',
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()


# Récupérer l'importance des features
# Récupérer le modèle du classifieur après l'entraînement
model = pipeline.named_steps['classifieur']

# Accéder aux importances des features
importances = model.feature_importances_


# Récupérer les noms des features après transformation
# Utiliser l'encodeur OneHotEncoder pour gérer les variables catégorielles en
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la tr

```

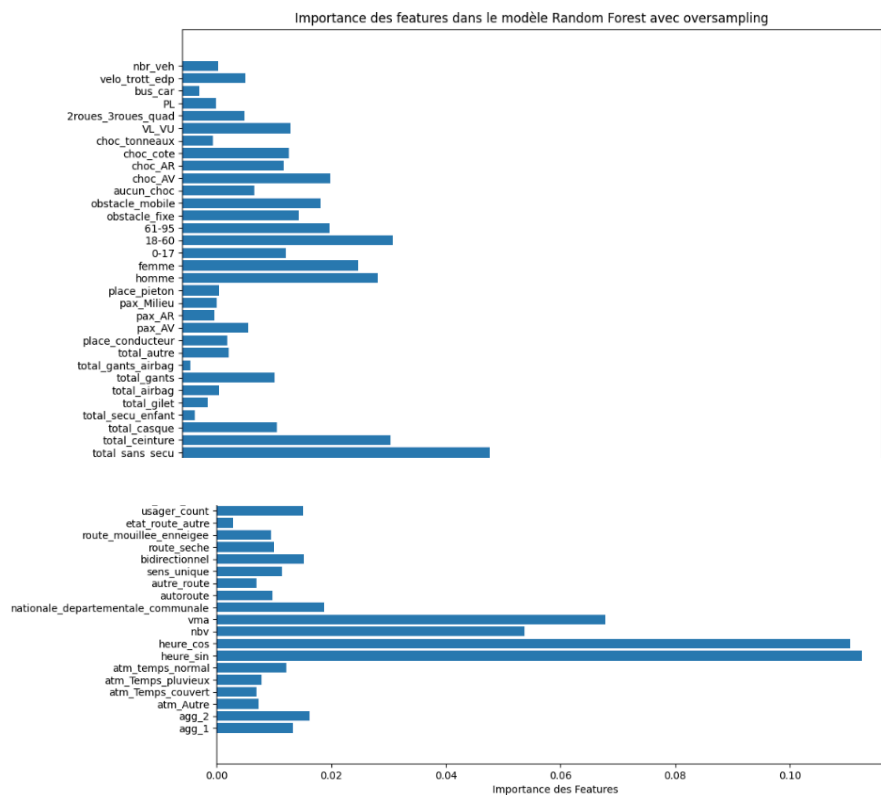
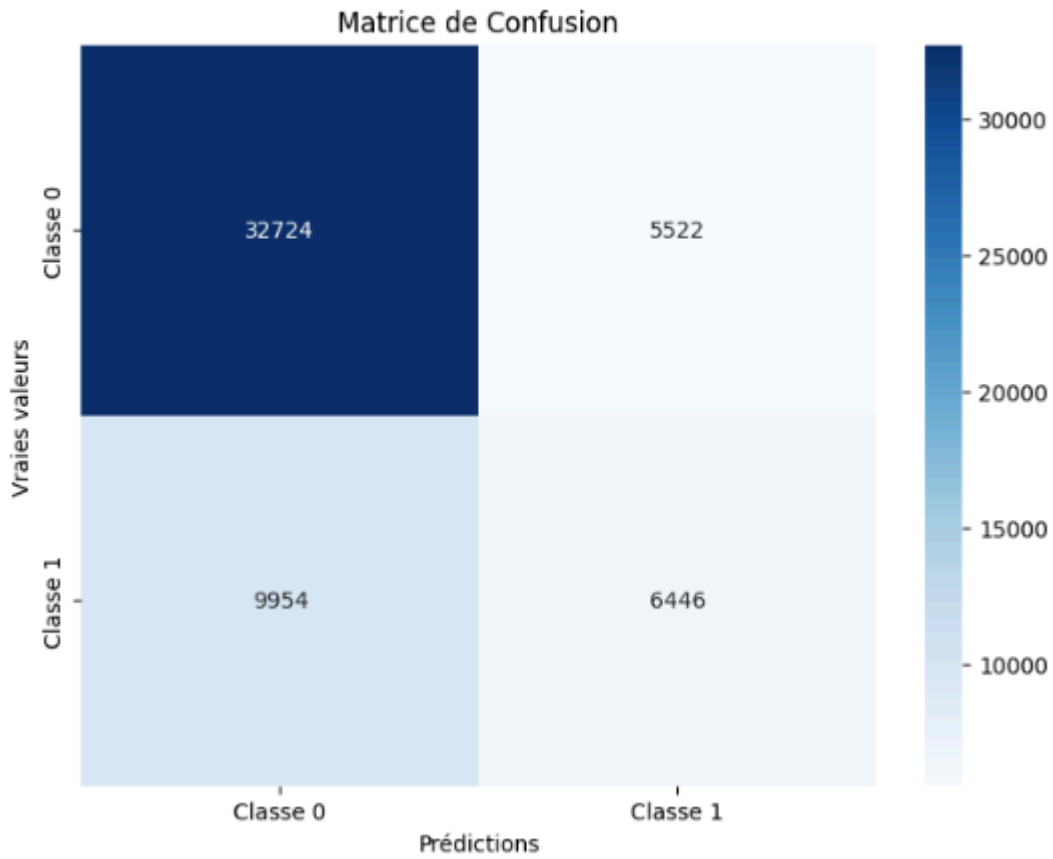
```
# Les noms des features après transformation
features = np.concatenate([cat_columns, cyclical_columns, passthrough_features])

# Tracer l'importance des features
plt.figure(figsize=(12, 12))
plt.barh(features, importances)
plt.xlabel("Importance des Features")
plt.title("Importance des features dans le modèle Random Forest avec oversampling")
plt.show()
```

```
Accuracy du modèle : 0.72
Précision : 0.70
Rappel : 0.72
F1-score : 0.70
```

Rapport de Classification :				
	precision	recall	f1-score	support
sans_urgence	0.77	0.86	0.81	38246
urgence	0.54	0.39	0.45	16400
accuracy			0.72	54646
macro avg	0.65	0.62	0.63	54646
weighted avg	0.70	0.72	0.70	54646





Recall classe 1 (urgent) : 0.39  
F1\_Score classe 1 (urgent): 0.45

## Random Forest avec les variables les plus importantes: vma, nbv, heure, total\_sans\_secu, nationale\_communale\_departementale

```
# Préparation des données
X = df.drop(['Num_Acc', 'total_ceinture', 'total_casque',
            'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
            'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
            'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
            '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
            'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
            '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh', 'autoroute',
            'sens_unique', 'bidirectionnel', 'route_seche', 'atm',
            'route_mouillee_enneigee', 'etat_route_autre', 'usager_count', 'lat', 'long', 'jc
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
```

```

# Liste des variables catégorielles et cycliques
categorical_features = ['agg'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'total_sans_secu'] # À laisser inchangées

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifieur', RandomForestClassifier(n_estimators=100, random_state=42))]

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracv du modèle : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

```

```

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cl
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

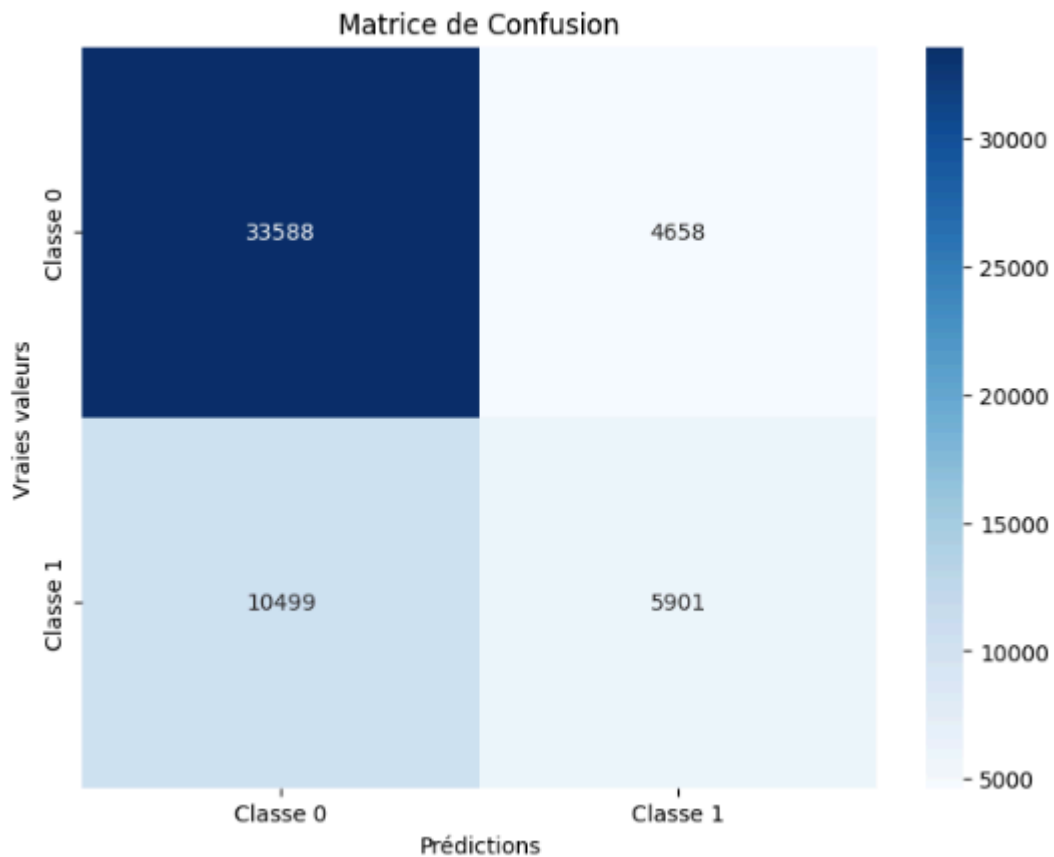
```

Accuracy du modèle : 0.72
Précision : 0.70
Rappel : 0.72
F1-score : 0.70

Rapport de Classification :

```

	precision	recall	f1-score	support
sans_urgence	0.76	0.88	0.82	38246
urgence	0.56	0.36	0.44	16400
accuracy			0.72	54646
macro avg	0.66	0.62	0.63	54646
weighted avg	0.70	0.72	0.70	54646



Recall classe 1 (urgent) : 0.36

F1\_Score classe 1 (urgent): 0.44

Les scores avec quelques variables sont inférieurs à ceux avec toutes les variables. Nous poursuivrons avec toutes les variables.

## Random Forest avec oversampling

```
# Préparation des données
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int']
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self
```

```

def transform(self, X):
    X = X.astype(float) # Assurer que les valeurs sont numériques
    X_sin = np.sin(2 * np.pi * X / self.period)
    X_cos = np.cos(2 * np.pi * X / self.period)
    return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Appliquer SMOTE dans la pipeline (avant l'entraînement du modèle)
smote = SMOTE(random_state=42, sampling_strategy = 'auto', k_neighbors = 5)

# Définir la pipeline complète avec SMOTE
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('smote', smote), # Oversampling via SMOTE
    ('classif', RandomForestClassifier(n_estimators=100, random_state=42))
])

```

```

])

# Entraîner le modèle avec oversampling
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle avec oversampling : {accuracy:.2f}")

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion

```

```

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cl
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# Récupérer l'importance des features
# Récupérer le modèle du classifieur après l'entraînement
model = pipeline.named_steps['classifieur']

# Accéder aux importances des features
importances = model.feature_importances_

# Récupérer les noms des features après transformation
# Utiliser l'encodeur OneHotEncoder pour gérer les variables catégorielles en
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la tr

# Les noms des features après transformation
features = np.concatenate([cat_columns, cyclical_columns, passthrough_featu

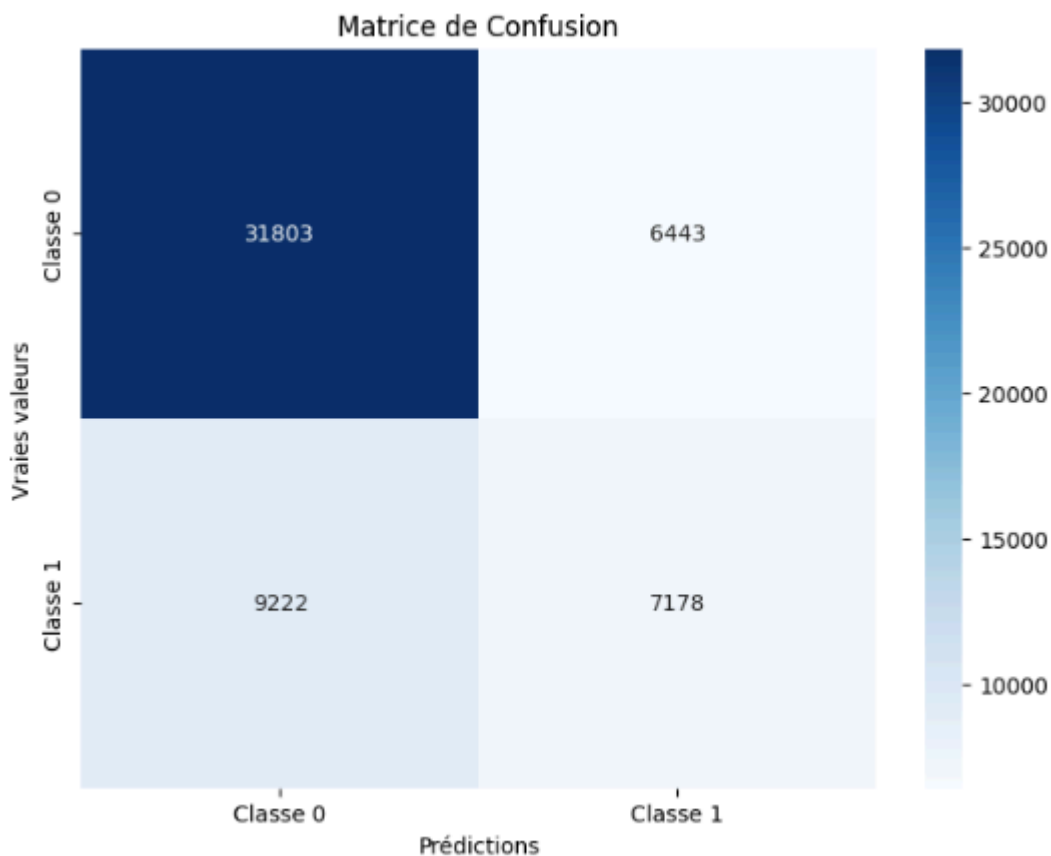
# Tracer l'importance des features
plt.figure(figsize=(12, 12))
plt.barh(features, importances)
plt.xlabel("Importance des Features")
plt.title("Importance des features dans le modèle Random Forest avec oversam
plt.show()

```



Accuracy du modèle avec oversampling : 0.71  
 Accuracy du modèle : 0.71  
 Précision : 0.70  
 Rappel : 0.71  
 F1-score : 0.71

Rapport de Classification :				
	precision	recall	f1-score	support
sans_urgence	0.78	0.83	0.80	38246
urgence	0.53	0.44	0.48	16400
accuracy			0.71	54646
macro avg	0.65	0.63	0.64	54646
weighted avg	0.70	0.71	0.71	54646



Recall classe 1 (urgent) : 0.44

F1\_Score classe 1 (urgent): 0.48

On constate une nette amélioration des scores avec l'oversampling.

## Random Forest avec Undersampling

```

# Préparation des données
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int']
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

```

```

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Appliquer RandomUnderSampler dans la pipeline (avant l'entraînement du modèle)
under_sampler = RandomUnderSampler(random_state=42)

# Définir la pipeline complète avec undersampling
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('under_sampler', under_sampler), # Undersampling via RandomUnderSampler
    ('classifieur', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Entraîner le modèle avec undersampling
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accurac   du mod  le avec undersampling : {accuracy:.2f}")

# Pr  cision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les r  sultats d  taill  s
print(f"Pr  cision : {precision:.2f}")

```

```

print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cl
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# Récupérer l'importance des features
# Récupérer le modèle du classifieur après l'entraînement
model = pipeline.named_steps['classifieur']

# Accéder aux importances des features
importances = model.feature_importances_

# Récupérer les noms des features après transformation
# Utiliser l'encodeur OneHotEncoder pour gérer les variables catégorielles en
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la tr

# Les noms des features après transformation
features = np.concatenate([cat_columns, cyclical_columns, passthrough_featu

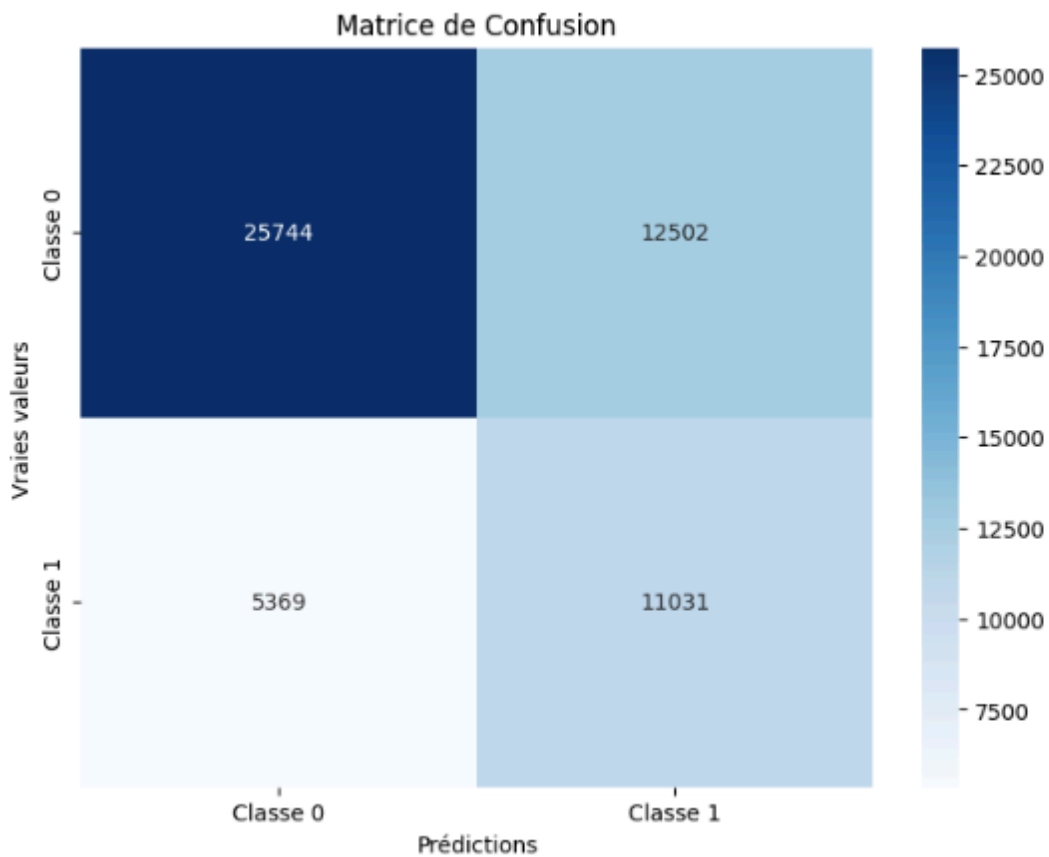
# Tracer l'importance des features
plt.figure(figsize=(12, 12))
plt.barh(features, importances)
plt.xlabel("Importance des Features")
plt.title("Importance des features dans le modèle Random Forest avec unders

```

Accuracy du modèle avec undersampling : 0.67  
 Précision : 0.72  
 Rappel : 0.67  
 F1-score : 0.69

Rapport de Classification :

	precision	recall	f1-score	support
sans_urgence	0.83	0.67	0.74	38246
urgence	0.47	0.67	0.55	16400
accuracy			0.67	54646
macro avg	0.65	0.67	0.65	54646
weighted avg	0.72	0.67	0.69	54646



Recall classe 1 (urgent) : 0.67

F1\_Score classe 1 (urgent): 0.55

Le modèle avec undersampling est très performant

**Grid search pour optimiser paramètres du modèle avec toutes les variables. Utilisation de**

## class\_weight='balanced' pour gérer le déséquilibre des classes

```
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
    'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
```

```

'2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

# 📌 **Pipeline de preprocessing**
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)
], remainder='passthrough')

# 📌 **Modèle de classification**
classifieur = RandomForestClassifier(random_state=42, class_weight='balanced')

# 📌 **Pipeline principal avec imblearn Pipeline**
pipeline = Pipeline([
    ('preprocessor', preprocessor), # Étape de transformation
    ('classifieur', classifieur) # Modèle final
])

# 📌 **Définition de la grille de recherche**
param_grid = {

    'classifieur__n_estimators': [50, 100, 200], # Nombre d'arbres
    'classifieur__max_depth': [10, 20, None], # Profondeur maximale
    'classifieur__class_weight': ['balanced', None] # Tester 'balanced' et None
}

# 📌 **Lancer la GridSearchCV**
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='f1_weighted')
grid_search.fit(X_train, y_train)

# 📌 **Afficher les meilleurs paramètres**
print("Meilleurs paramètres:", grid_search.best_params_)

# 📌 **Évaluation du modèle optimal**
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

```

```

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))

# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()

```

Meilleurs paramètres: {'classifier\_\_class\_weight': 'balanced', 'classifier\_\_max\_depth': 20, 'classifier\_\_n\_estimators': 200}

```

📊 **Performance du modèle optimal**
Accuracy: 0.72
Precision: 0.73
Recall: 0.72
F1-score: 0.73

```

```

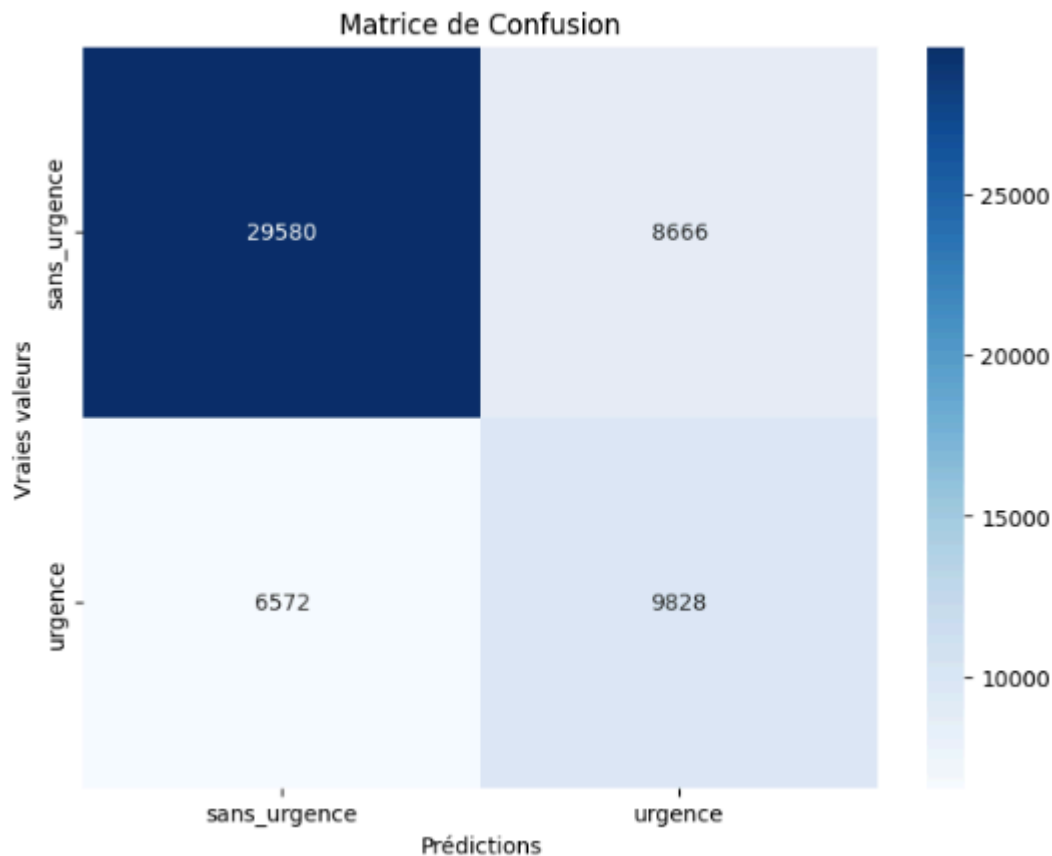
Rapport de Classification:
              precision    recall  f1-score   support

sans_urgence    0.82      0.77      0.80     38246
urgence         0.53      0.60      0.56     16400

   accuracy          0.72          0.72          0.72     54646
  macro avg          0.67          0.69          0.68     54646
weighted avg          0.73          0.72          0.73     54646

```





Recall classe 1 (urgent) : 0.60

F1\_Score classe 1 (urgent): 0.56

Ces paramètres sont moins performants que l'undersampling pour le Recall, et équivalents pour le F1\_Score.

## GridSearch avec les variables les plus importantes pour optimiser les paramètres du modèle

```
# Préparation des données
X = df.drop(['Num_Acc', 'total_ceinture', 'total_casque',
            'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
            'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
            'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
            '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
            'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
            '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh', 'autoroute',
            'sens_unique', 'bidirectionnel', 'route_seche', 'atm',
            'route_mouillee_enneigee', 'etat_route_autre', 'usager_count', 'lat', 'long', 'jour'], axis=1)
y = df['gravité_accident']
```

```

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 📌 **Définition des variables**
categorical_features = ['agg']
cyclical_features = ['heure']
passthrough_features = ['nbv', 'vma']

# 📌 **Pipeline de preprocessing**
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)
], remainder='passthrough')

# 📌 **Modèle de classification**
classifier = RandomForestClassifier(random_state=42)

# 📌 **Pipeline principal avec imblearn Pipeline**
pipeline = Pipeline([
    ('preprocessor', preprocessor), # Étape de transformation

```

```

('classifier', classifier) # Modèle final
])

# 📌 **Définition de la grille de recherche**
param_grid = {

    'classifier__n_estimators': [50, 100, 200], # Nombre d'arbres
    'classifier__max_depth': [10, 20, None], # Profondeur maximale
}

# 📌 **Lancer la GridSearchCV**
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='f1_weighted')
grid_search.fit(X_train, y_train)

# 📌 **Afficher les meilleurs paramètres**
print("Meilleurs paramètres:", grid_search.best_params_)

# 📌 **Évaluation du modèle optimal**
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')


print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))

# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)

```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()
```

Meilleurs paramètres: {'classifier\_\_max\_depth': 10, 'classifier\_\_n\_estimators': 50}

 **\*\*Performance du modèle optimal\*\***

Accuracy: 0.73

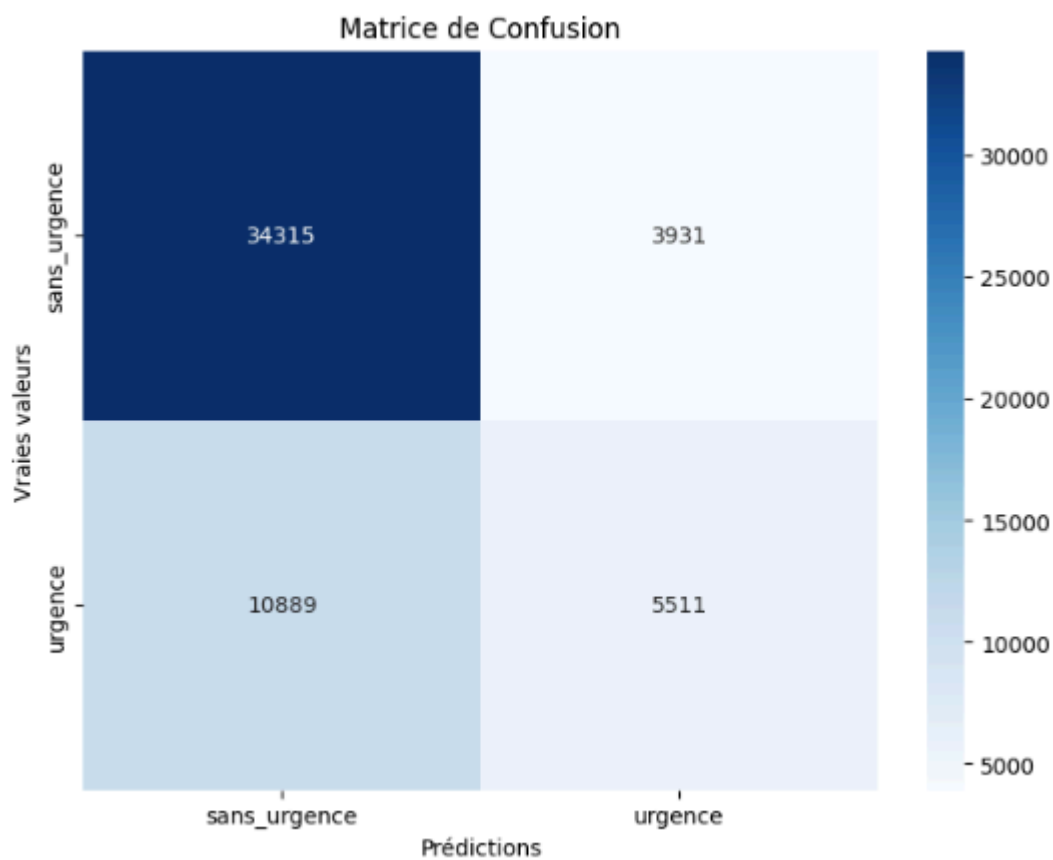
Precision: 0.71

Recall: 0.73

F1-score: 0.70

Rapport de Classification:

	precision	recall	f1-score	support
sans_urgence	0.76	0.90	0.82	38246
urgence	0.58	0.34	0.43	16400
accuracy			0.73	54646
macro avg	0.67	0.62	0.62	54646
weighted avg	0.71	0.73	0.70	54646



Recall classe 1 (urgent) : 0.34  
F1\_Score classe 1 (urgent): 0.43

Scores très mauvais

## Random Forest avec métrique pondérée pour optimiser le Recall

```
from sklearn.metrics import recall_score

# Préparation des données
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int']
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
```

```

'total_sans_secu', 'total_ceinture', 'total_casque',
'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
'18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
'2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laiss

preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Préparation des données
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
            'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

from sklearn.preprocessing import LabelEncoder

# Encoder les labels en entiers
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['gravité_accident'])

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

```

```

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_en',
                        'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'total',
                        'pax_AV', 'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme',
                        'obstacle_fixe', 'obstacle_mobile', 'aucun_choc', 'choc_AV', 'choc',
                        'VL_VU', '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', '']

# Définir les différentes pondérations à tester (déjà définies dans ton code précédent)
class_weights_list = [
    {0: 1, 1: 2},
    {0: 1, 1: 3},
    {0: 1, 1: 5},
    {0: 1, 1: 10},
]

# Classe 0: sans_urgence
# Classe 1: urgence
# Définir une fonction pour la précision pondérée
def weighted_f1_score(y_true, y_pred, class_weights):
    # Calculer la précision pour chaque classe
    f1_per_class = f1_score(y_true, y_pred, average=None)

    # Associer les poids aux classes dans le même ordre que dans precision_per_class
    weights = [class_weights.get(i, 1) for i in range(len(f1_per_class))]

    # Calculer la précision pondérée en fonction des poids
    weighted_f1 = np.dot(f1_per_class, weights) / sum(weights)
    return weighted_f1

```

```

# Définir la pipeline complète
pipeline = Pipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('classifier', RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42))
])

# Pour chaque configuration de poids, effectuer une validation croisée et calculer la précision pondérée
results = {}

for class_weights in class_weights_list:
    # Utiliser cross_val_score pour évaluer la performance avec les poids de classe
    f1_scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='f1_weighted', n_jobs=-1)

    # Enregistrer la moyenne des scores de précision pondérée
    results[str(class_weights)] = np.mean(f1_scores)

# Afficher les résultats pour chaque configuration de pondération
for class_weights, score in results.items():
    print(f"Configuration des poids {class_weights} → Précision pondérée : {score}")

# Trouver la configuration qui donne la meilleure précision pondérée
best_class_weights = max(results, key=results.get)
print(f"\nMeilleure configuration de pondération : {best_class_weights}")

# Créer la pipeline en utilisant les poids optimaux dans le classificateur Random Forest
pipeline_with_optimal_weights = Pipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('classifier', RandomForestClassifier(
        n_estimators=50,
        max_depth=10,
        random_state=42,
        class_weight=eval(best_class_weights) # Appliquer les poids ici
    ))
])

# Entraîner le modèle avec les poids optimaux
pipeline_with_optimal_weights.fit(X_train, y_train)

```



```

# Faire des prédictions sur l'ensemble de test
y_pred = pipeline_with_optimal_weights.predict(X_test)

# Evaluation de la performance du modèle

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall_optimal = recall_score(y_test, y_pred, average='weighted')
f1_score = f1_score(y_test, y_pred, average='weighted')


print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall_optimal: {recall:.2f}")
print(f"F1-score: {f1_score:.2f}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))

# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()

```

Configuration des poids {0: 1, 1: 2} -> Précision pondérée : 0.5588953910759638  
 Configuration des poids {0: 1, 1: 3} -> Précision pondérée : 0.5254302035131241  
 Configuration des poids {0: 1, 1: 5} -> Précision pondérée : 0.4919650159502845  
 Configuration des poids {0: 1, 1: 10} -> Précision pondérée : 0.4615421181658849

Meilleure configuration de pondération : {0: 1, 1: 2}

 **\*\*performance du modèle optimal\*\***

Accuracy: 0.72

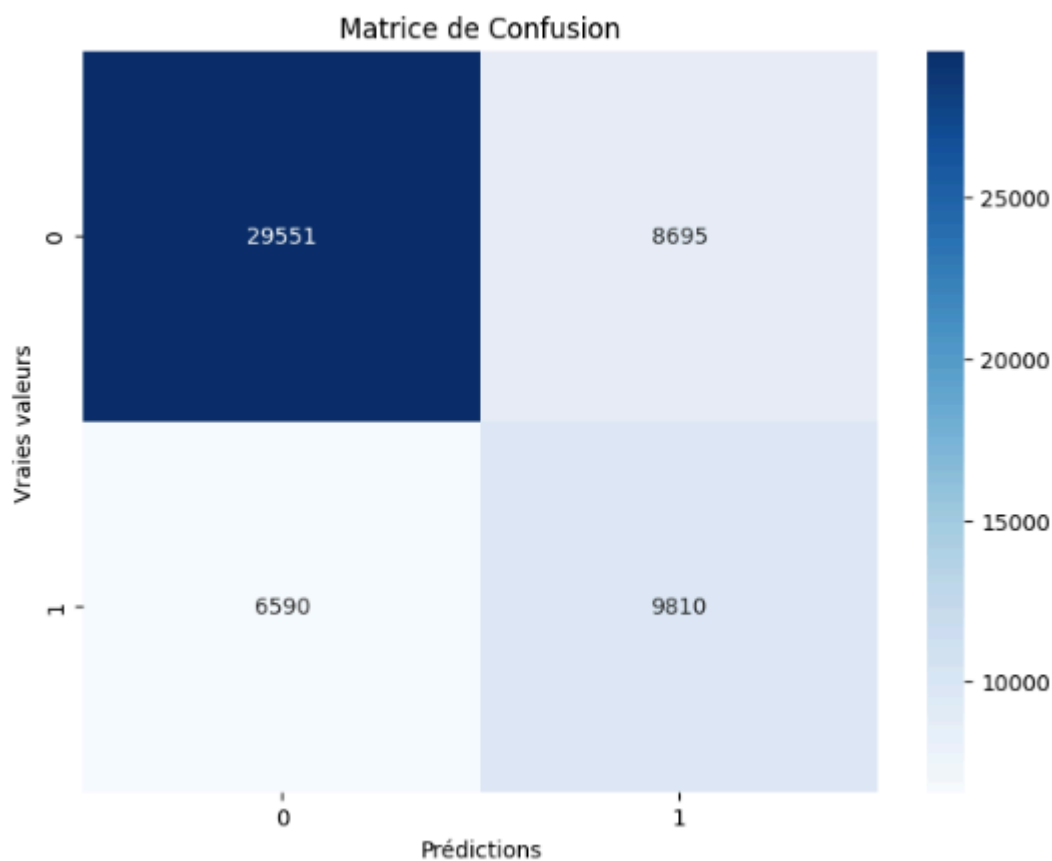
Precision: 0.73

Recall\_optimal: 0.72

F1-score: 0.72

Rapport de Classification:

	precision	recall	f1-score	support
0	0.82	0.77	0.79	38246
1	0.53	0.60	0.56	16400
accuracy			0.72	54646
macro avg	0.67	0.69	0.68	54646
weighted avg	0.73	0.72	0.72	54646



Recall classe 1 (urgent) : 0.60

F1\_Score classe 1 (urgent): 0.56

## Grid search avec scoring personnalisé pour optimiser F1\_score classe urgence

```
from sklearn.metrics import make_scorer, f1_score, accuracy_score, precision.  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model_selection import train_test_split, GridSearchCV  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.compose import ColumnTransformer  
from sklearn.pipeline import Pipeline  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.base import BaseEstimator, TransformerMixin  
from sklearn.metrics import confusion_matrix  
  
# Définition de la classe pour la transformation de l'heure  
class CyclicalFeatures(BaseEstimator, TransformerMixin):  
    """Transforme une colonne de type heure en variables cycliques sin et cos.  
    def __init__(self, period=24):  
        self.period = period  
  
    def fit(self, X, y=None):  
        return self  
  
    def transform(self, X):  
        X = X.astype(float) # Assurer que les valeurs sont numériques  
        X_sin = np.sin(2 * np.pi * X / self.period)  
        X_cos = np.cos(2 * np.pi * X / self.period)  
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos  
  
# Séparer X et y  
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',  
y = df['gravité_accident']
```

```

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# 📌 **Pipeline de preprocessing**
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)
], remainder='passthrough')

# 📌 **Définir un scorer personnalisé pour optimiser le F1-score de la classe 1**
def class_specific_f1(y_true, y_pred, class_index=1):
    return f1_score(y_true, y_pred, labels=[class_index], average='micro')

# Crée un scorer personnalisé basé sur le F1-score pour la classe 1
custom_f1_scorer = make_scorer(class_specific_f1, class_index=1)

# 📌 **Modèle de classification**
classifieur = RandomForestClassifier(random_state=42, class_weight='balanced')

# 📌 **Pipeline principal avec imblearn Pipeline**

```

```

pipeline = Pipeline([
    ('preprocessor', preprocessor), # Étape de transformation
    ('classifieur', classifieur) # Modèle final
])

# 📌 **Définition de la grille de recherche**
param_grid = {
    'classifieur__n_estimators': [50, 100, 200], # Nombre d'arbres
    'classifieur__max_depth': [10, 20, None], # Profondeur maximale
    'classifieur__class_weight': ['balanced', None] # Tester 'balanced' et None
}

# 📌 **Lancer la GridSearchCV avec le scorer personnalisé**
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring=custom_f1_scorer)
grid_search.fit(X_train, y_train)

# 📌 **Afficher les meilleurs paramètres**
print("Meilleurs paramètres:", grid_search.best_params_)

# 📌 **Évaluation du modèle optimal**
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Calcul des métriques
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("\n📊 **Performance du modèle optimal**")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
print("\nRapport de Classification:")
print(classification_report(y_test, y_pred))

```

```
# 📌 **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()
```

Meilleurs paramètres: {'classifier\_\_class\_weight': 'balanced', 'classifier\_\_max\_depth': 10, 'classifier\_\_n\_estimators': 50}

📊 \*\*Performance du modèle optimal\*\*

Accuracy: 0.70

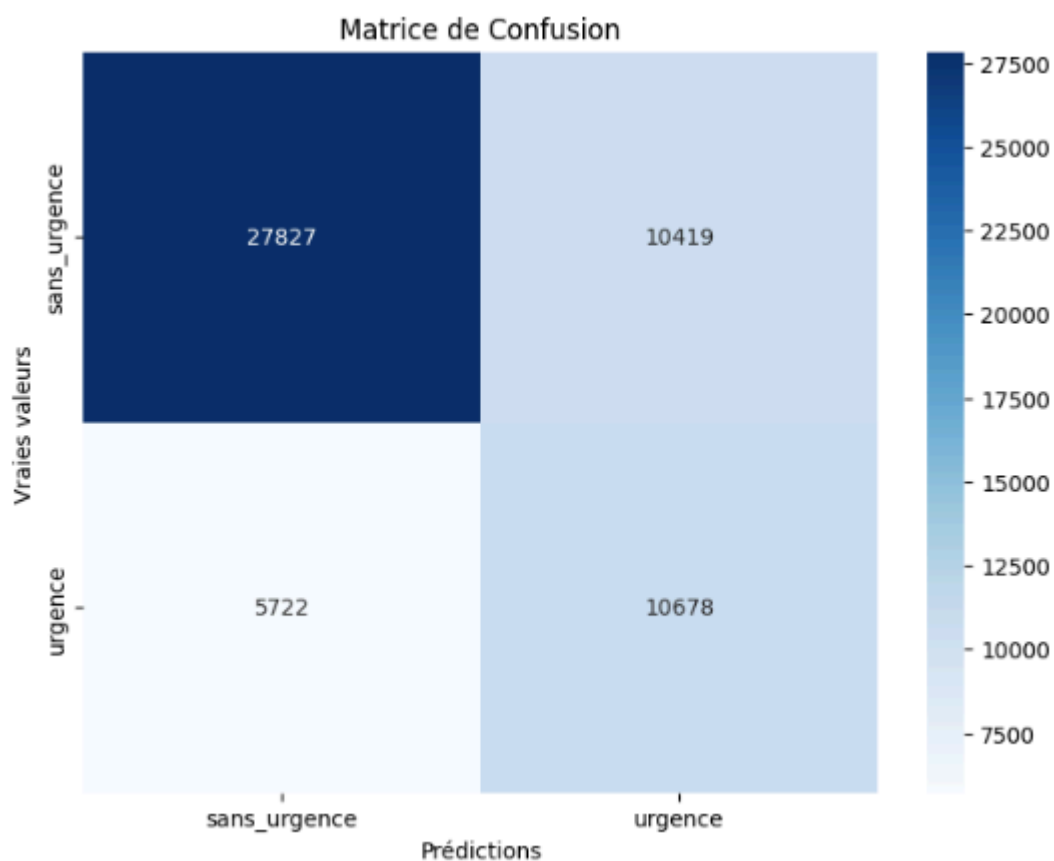
Precision: 0.73

Recall: 0.70

F1-score: 0.71

Rapport de Classification:

	precision	recall	f1-score	support
sans_urgence	0.83	0.73	0.78	38246
urgence	0.51	0.65	0.57	16400
accuracy			0.70	54646
macro avg	0.67	0.69	0.67	54646
weighted avg	0.73	0.70	0.71	54646



Recall classe 1 (urgent) : 0.65  
F1\_Score classe 1 (urgent): 0.57

**Conclusion du modèle Random Forest à deux classes:**  
Le modèle avec undersampling est le plus performant pour le Recall(0.67), tandis que le modèle avec les paramètres {'classifier\_\_class\_weight': 'balanced', 'classifier\_\_max\_depth': 10, 'classifier\_\_n\_estimators': 50} est le plus performant pour le F1\_Score (0.57)

## VIII. Modèle 2 classes Régression Logistique

### Régression logistique simple

```
# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int'
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos
```

```

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec la régression logistique
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classif', LogisticRegression(max_iter=1000, random_state=42))] # Rég

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

```



```

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['sa
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

```

Accuracy du modèle : 0.73
Précision : 0.71
Rappel : 0.73
F1-score : 0.70

Rapport de Classification :

```

	precision	recall	f1-score	support
sans_urgence	0.76	0.91	0.83	38246
urgence	0.60	0.32	0.41	16400
accuracy			0.73	54646
macro avg	0.68	0.61	0.62	54646
weighted avg	0.71	0.73	0.70	54646

## Régression logistique avec oversampling

```

# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc','lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
y = df['gravité_accident']

# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc','lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

```

```

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
    'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
    '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec RandomUnderSampler (undersampling) et
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression
])

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle avec undersampling : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')

```

```

recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['bl', 'br', 'br', 'br'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)

```

```

return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec SMOTE (oversampling) et la régression log
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('smote', SMOTE(random_state=42)), # Oversampling avec SMOTE
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression
])

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

```

```

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle avec oversampling : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['bl', 'br', 'br', 'br', 'br', 'br', 'br', 'br'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

```

Accuracy du modèle avec oversampling : 0.68
Précision : 0.72
Rappel : 0.68
F1-score : 0.69

Rapport de Classification :

```

	precision	recall	f1-score	support
0	0.83	0.68	0.75	38246
1	0.47	0.68	0.56	16400
accuracy			0.68	54646
macro avg	0.65	0.68	0.65	54646
weighted avg	0.72	0.68	0.69	54646

## Régression logistique avec undersampling

```

# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc','lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
y = df['gravité_accident']

# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc','lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int',
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec RandomUnderSampler (undersampling) et
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression
])

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuraciy du modèle avec undersampling : {accuracy:.2f}")

```



```

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['bl', 'br', 'br', 'br'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)

```

```

X_cos = np.cos(2 * np.pi * X / self.period)
return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
                        'sens_unique', 'bidirectionnel', 'route_seche',
                        'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
                        'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
                        'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
                        'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
                        '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
                        'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
                        '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec RandomUnderSampler (undersampling) et
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression
])

# Entraîner le modèle
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

```

```

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle avec undersampling : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['bl', 'br', 'br', 'br'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

```

Accuracy du modèle avec undersampling : 0.68
Précision : 0.72
Rappel : 0.68
F1-score : 0.69

Rapport de Classification :

```

	precision	recall	f1-score	support
sans_urgence	0.83	0.68	0.75	38246
urgence	0.48	0.67	0.56	16400
accuracy			0.68	54646
macro avg	0.65	0.68	0.65	54646
weighted avg	0.72	0.68	0.69	54646

## Conclusion:

Nous obtenons de très bons scores, meilleurs qu'avec Random Forest, avec l'oversampling.

On a alors le Recall de la classe urgent qui s'élève à 0.68 et le F1\_Score à 0.56

## IX. Modèle 2 classes KNN

### KNN avec scoring pour optimiser recall

```

from google.colab import drive
import pandas as pd
import os
from io import StringIO
# Monter Google Drive
drive.mount('/content/drive', force_remount= True) #force_remount = True pe
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Data:
df['gravité_accident'] = df['gravité_accident'].replace({2: 'sans_urgence', 3: 'u
df['gravité_accident'].value_counts()

import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin

```

```

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import matplotlib.pyplot as plt
import seaborn as sns

# Préparation des données (Exemple de df à adapter)
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int'
y = df['gravité_accident']

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',

```

```

'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
'18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
'2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À lais

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Définir la pipeline complète avec KNN
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classif', KNeighborsClassifier(n_neighbors=5))]) # KNN avec 5 voisins

# Créer le scorer pour optimiser le recall
recall_scorer = make_scorer(recall_score, average='weighted')

# Recherche des hyperparamètres avec GridSearchCV
param_grid = {'classif__n_neighbors': [3, 5, 7, 9, 11]} # Exemple de grid de \

# GridSearchCV pour optimiser le recall
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring=recall_score)
grid_search.fit(X_train, y_train)

# Meilleurs hyperparamètres et meilleure score
print(f"Meilleurs paramètres : {grid_search.best_params_}")
print(f"Meilleur recall : {grid_search.best_score_:.2f}")

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

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

```

```

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")
print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['sa
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

```

Meilleurs paramètres : {'classifier__n_neighbors': 3}
Meilleur recall : nan
Accuracy du modèle : 0.70
Précision : 0.68
Rappel : 0.70
F1-score : 0.69

```

```

Rapport de Classification :

```

	precision	recall	f1-score	support
sans_urgence	0.77	0.81	0.79	38246
urgence	0.49	0.42	0.45	16400
accuracy			0.70	54646
macro avg	0.63	0.62	0.62	54646
weighted avg	0.68	0.70	0.69	54646

Les scores sont mauvais, bien loin de ceux de Random Forest ou de Logistic Régression

## X. Modèle 2 classes XGBoost

### XGBOOST avec l'argument classweight pour gérer le déséquilibre des classes

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEnc
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils.class_weight import compute_sample_weight
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):
    """Transforme une colonne de type heure en variables cycliques sin et cos.
    def __init__(self, period=24):
        self.period = period

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.astype(float) # Assurer que les valeurs sont numériques
        X_sin = np.sin(2 * np.pi * X / self.period)
        X_cos = np.cos(2 * np.pi * X / self.period)
        return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparation des variables d'entrée et de sortie
```



```

X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué',
            'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

# Encoder les classes texte en entiers
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y) # Transforme 'sans_urgence' et 'urgence' en entiers

# Séparation en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Variables à encoder
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillee_en',
                        'usager_count', 'total_sans_secu', 'total_ceinture', 'total_casque',
                        'total_gilet', 'total_airbag', 'total_gants', 'total_gants_airbag', 'total',
                        'place_conducteur', 'pax_AV', 'pax_AR', 'pax_Milieu', 'place_piet',
                        '0-17', '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun',
                        'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU', '2roues_3roues',
                        'velo_trott_edp', 'nbr_veh'] # À laisser inchangées

# Transformation des données
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features),
    ('scaler', StandardScaler(), passthrough_features) # Normalisation pour XG
])

# Calcul des poids des classes
class_weights = compute_sample_weight(class_weight='balanced', y=y_train)

# Créer et entraîner le modèle XGBoost
model = XGBClassifier(
    objective='multi:softmax', # Classification multiclass
    num_class=3, # Nombre de classes
    eval_metric='mlogloss', # Log loss

```

```

    use_label_encoder=False, # Suppression d'un avertissement lié à XGBoost
    random_state=42
)

# Appliquer la transformation de prétraitement sur les données d'entraînement
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# Entraîner le modèle avec les poids de classes
model.fit(X_train_transformed, y_train, sample_weight=class_weights)

# Prédiction
y_pred = model.predict(X_test_transformed)

# Évaluation du modèle
accuracy = accuracy_score(y_test, y_pred) # Les labels sont déjà sous forme
print(f"Accuracy du modèle : {accuracy:.2f}")

# Rapport de classification détaillé
print("\nRapport de classification :")
print(classification_report(y_test, y_pred))

# Visualisation de la matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.xlabel('Prédictions')
plt.ylabel('Vraies Classes')
plt.title('Matrice de Confusion')
plt.show()

```

```

Accuracy du modèle : 0.70

Rapport de classification :

```

	precision	recall	f1-score	support
0	0.84	0.70	0.77	38200
1	0.50	0.70	0.59	16446
accuracy			0.70	54646
macro avg	0.67	0.70	0.68	54646
weighted avg	0.74	0.70	0.71	54646

Il s'agit là des meilleurs scores obtenus pour la classe urgent depuis le début de nos essais

## XGBOOST avec oversampling

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from xgboost import XGBClassifier

# Préparation des données
X = df.drop(['Num_Acc', 'lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int'
y = df['gravité_accident']

from sklearn.preprocessing import LabelEncoder

# Encoder les labels en entiers
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['gravité_accident']) # Convertit 'sans_urge

# Définition de la classe pour la transformation de l'heure
class CyclicalFeatures(BaseEstimator, TransformerMixin):

```

```

"""Transforme une colonne de type heure en variables cycliques sin et cos.
def __init__(self, period=24):
    self.period = period

def fit(self, X, y=None):
    return self

def transform(self, X):
    X = X.astype(float) # Assurer que les valeurs sont numériques
    X_sin = np.sin(2 * np.pi * X / self.period)
    X_cos = np.cos(2 * np.pi * X / self.period)
    return np.c_[X_sin, X_cos] # Retourne un array 2D avec sin et cos

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Liste des variables catégorielles et cycliques
categorical_features = ['agg', 'atm'] # Encodage
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale', 'departementale', 'communale',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillee_enneigee', 'etat_route_autre', 'usager_count',
    'total_sans_secu', 'total_ceinture', 'total_casque',
    'total_secu_enfant', 'total_gilet', 'total_airbag', 'total_gants',
    'total_gants_airbag', 'total_autre', 'place_conducteur', 'pax_AV',
    'pax_AR', 'pax_Milieu', 'place_pieton', 'homme', 'femme', '0-17',
    '18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc',
    'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneaux', 'VL_VU',
    '2roues_3roues_quad', 'PL', 'bus_car', 'velo_trott_edp', 'nbr_veh'] # À laisser

# Appliquer le ColumnTransformer avec `remainder='passthrough'`
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ('cyclical', CyclicalFeatures(), cyclical_features)], remainder='passthrough')

# Appliquer SMOTE dans la pipeline (avant l'entraînement du modèle)
smote = SMOTE(random_state=42, sampling_strategy = 'auto', k_neighbors = 5)

```

```

# Définir la pipeline complète avec SMOTE
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Transformation des variables
    ('smote', smote), # Oversampling via SMOTE
    ('classifier', XGBClassifier(objective='multi:softmax', num_class=3, eval_me
)])

# Entraîner le modèle avec oversampling
pipeline.fit(X_train, y_train)

# Prédiction
y_pred = pipeline.predict(X_test)

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle avec oversampling : {accuracy:.2f}")

# Évaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy du modèle : {accuracy:.2f}")

# Précision, Rappel, F1-score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)

# Rapport de classification
class_report = classification_report(y_test, y_pred)

# Afficher les résultats détaillés
print(f"Précision : {precision:.2f}")

```

```

print(f"Rappel : {recall:.2f}")
print(f"F1-score : {f1:.2f}")
print("\nRapport de Classification :")
print(class_report)

# Matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cl
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

# Récupérer l'importance des features
# Récupérer le modèle du classifieur après l'entraînement
model = pipeline.named_steps['classifieur']

# Accéder aux importances des features
importances = model.feature_importances_

# Récupérer les noms des features après transformation
# Utiliser l'encodeur OneHotEncoder pour gérer les variables catégorielles en
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la tr

# Les noms des features après transformation
features = np.concatenate([cat_columns, cyclical_columns, passthrough_featu

# Tracer l'importance des features
plt.figure(figsize=(12, 12))
plt.barh(features, importances)
plt.xlabel("Importance des Features")
plt.title("Importance des features dans le modèle Random Forest avec oversam
plt.show()

```

```
Accuracy du modèle avec oversampling : 0.74
Accuracy du modèle : 0.74
Précision : 0.73
Rappel : 0.74
F1-score : 0.73
```

```
Rapport de Classification :
      precision    recall  f1-score   support

0         0.79        0.85        0.82     38246
1         0.58        0.47        0.52     16400

 accuracy          0.74     54646
  macro avg         0.68     0.66     0.67     54646
 weighted avg         0.73     0.74     0.73     54646
```

L'oversampling est moins performant dans notre cas pour le modèle XGBoost.

Conclusion:

Nous obtenons de très bons scores, meilleurs qu'avec Random Forest et Logistic Regression, avec l'argument `class_weight = balanced`.

On a alors le Recall de la classe urgent qui s'élève à 0.70 et le F1\_Score à 0.59.