

# Rapport final: Accidents routiers en France

**L'objectif de ce projet est de prédire la gravité des accidents routiers en France en se basant sur les données historiques recensées en France entre 2005 et 2023.**

Nous avons à notre disposition des datasets regroupant des données recueillies par l'unité des forces de l'ordre qui est intervenue sur le lieu de l'accident, ainsi que la description des bases de données annuelles.

Les fichiers sont séparés en 4 rubriques:

1. La rubrique CARACTERISTIQUES qui décrit les circonstances générales de l'accident
2. La rubrique LIEUX qui décrit le lieu principal de l'accident même si celui-ci s'est déroulé à une intersection
3. La rubrique VEHICULES impliqués
4. La rubrique USAGERS impliqués

Dans une première partie, nous préparerons un dataframe exploitable à partir de toutes les données à notre disposition, puis nous les analyserons, et enfin, nous prédirons la gravité des accidents de la route en France grâce à différents modèles de machine learning.

## **PLAN**

### **I. Nettoyage et analyse des données**

1. Découverte des données
2. Précision du sujet
3. Elaboration d'un dataframe unique de 2019 à 2023
4. Analyse des données
  - a. Conditions de l'accident
    - Date
    - Vitesse maximale autorisée
  - b. Usagers
    - Sexe
    - Age
    - Système de sécurité
  - c. Localisation de l'accident
    - Carte géographique de la répartition des accidents
    - Localisation agglomération/hors agglomération
  - d. Véhicules impliqués
    - Types de véhicules
    - Points de choc
  - e. Matrices de corrélation
    - Corrélation selon les conditions des usagers
    - Corrélation selon les conditions de la route
    - Corrélation selon les obstacles et types de chocs

### **II. Modélisation pour prédire la gravité d'un accident**

1. Feature engeneering
2. Meilleurs modèles 3 classes
  - a. Modèle 3 classes Random Forest
  - b. Modèle 3 classes Logistic Regression
  - c. Modèle 3 classes KNN
  - d. Modèle 3 classes XGBoost
3. Meilleurs modèles 2 classes
  - a. Modèle 2 classes Random Forest
  - b. Modèle 2 classes Logistic Regression
  - c. Modèle 2 classes KNN
  - d. Modèle 2 classes XGBoost
4. Tableau récapitulatif des résultats des modèle

### **III. Codes**

1. Partie nettoyage et analyse de données
2. Partie modélisation

**Annexes:** Document présentant les différentes variables

# I. Nettoyage et analyse des données

## 1. Découverte des données

Un premier coup d'oeil aux diverses bases de données et au document de description de ces données met en évidence:

- Une scission dans la manière de renseigner les données à partir de 2019 (exemples: NumAcc renseigné sous AccidentId, Les heures renseignée par tranches de 30 minutes et désormais renseignées à la minute près....), et des données plus complètes à partir de 2019.
- Le numero d'accident (Num\_Acc) est commun à toutes les rubriques.
- Beaucoup trop de lignes pour pouvoir fusionner toutes les rubriques et toutes les années en un seul dataframe (1 247 773 pour les lieux par exemple de 2005 à 2023).
- Des niveaux d'agrégation différents: pour les rubriques véhicules et usagers, plusieurs lignes correspondent à un même accident: une ligne par usager impliqué pour usagers, et une ligne par véhicule impliqué pour véhicules.
- Un nombre important de colonnes (56), avec pour certaines beaucoup de valeurs uniques.
- Nécessité de préciser notre variable cible: qu'entend-on par "gravité" qui peut être vue d'un point de vue du nombre de véhicules impliqués, du nombre d'usagers impliqués, du nombre de personnes tuées ?

## 2. Précision du sujet

Après réflexion collective, nous décidons de créer un modèle dont l'objectif serait de permettre à un centre d'appel de secours de déterminer le niveau de gravité d'un accident qui vient de se produire, afin de déclencher le nombre et le type de secours adapté. La variable cible est la gravité.

Dans cette optique, nous prenons deux décisions:

- Se concentrer sur les données de 2019 à 2023, qui sont les plus récentes, et renseignées de manière uniforme. Cela nous permettra de fusionner les 4 rubriques pour relier toutes les variables à notre variable cible, et d'avoir suffisamment de lignes (273226) pour construire notre modèle de machine learning.
- Supprimer les variables qui nous paraissent sans rapport avec la gravité.

Pour cela, nous concaténons les dataframe des années de 2019 à 2023 par rubrique, puisqu'au sein d'une même rubrique, ils ont tous le même format.

Nous supprimons les variables suivantes, que nous déterminons non pertinentes pour notre problématique: 'lum', 'com', 'adr', 'voie', 'v1', 'v2', 'circ', 'vosp', 'prof', 'pr', 'pr1', 'plan', 'lartpc', 'larrou', 'infra', 'situ', 'id\_vehicule', 'num\_veh', 'senc', 'occutc', 'id\_usager', 'trajet', 'locp', 'actp', 'etatp'

## 3. Elaboration d'un dataframe unique de 2019 à 2023

Dans le but de conserver un maximum de données des datasets initiaux, un travail est nécessaire sur chacune des tables avant de pouvoir merger le tout dans un dataset unique. Ceci en raison du niveau d'agrégation différent.

La table Usagers est celle ayant le niveau le plus fin d'agrégation et donc celle par laquelle nous commençons, suivie de la table véhicules.

Un premier travail consiste à étudier les différentes colonnes, et les valeurs uniques pour chacune d'elle, et déterminer si c'est une colonne que nous souhaitons conserver ou non (pertinente par rapport à notre objectif).

Un deuxième travail consiste à réfléchir sur des premiers regroupements possibles des valeurs uniques contenues

dans chacune des colonnes (par exemple: la colonne "surface" détaille initialement si la surface est sèche,mouillée, enneigée, boueuse, flaques d'eau, flaques d'huile,... que nous choisissons de regrouper en seulement 3 valeurs: route sèche, route mouillée/enneigée et autre état, selon la distribution initiale des valeurs).

Une fois ces deux premières étapes réalisées, un troisième travail consiste à transformer chacune de ces colonnes en nouvelles colonnes correspondant aux valeurs possibles, selon nos regroupements.

L'objectif de ce travail étant de n'avoir en résultat final qu'une seule ligne par accident, tout en ayant conservé un maximum d'informations.

Ainsi, là où nous avions initialement une table comme ceci par exemple avec 2 lignes pour un même accident:

**\*\*NumAcc | Surface | Condition Atmosphérique | Type de véhicule | Catégorie Usager\*\***

001 | mouillée | brouillard | Poids Lourd | conducteur

001 | mouillée | brouillard | Moto | Conducteur

Nous avons désormais une ligne par accident et les informations à disposition dans de nouvelles colonnes ( ici à titre d'exemple):

**\*\*NumAcc | Surf\_Mouillée\_enneigee |Surf\_sèche | Brouillard | Pluie| Poids\_Lourds | 2roues\_3roues | Cat\_conducteur | Cat\_passager |\*\***

001 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 0

Nous décidons par ailleurs de requalifier la gravité de l'accident, là où nous avions initialement une gravité pour chaque usager ( indemne, blessé léger, blessé hospitalisé ou tué)

Nous conservons la gravité la plus élevée, ainsi nous avons 3 niveaux de gravité différents, blessé léger (2), blessé hospitalisé (3), et mortel (4).

(Nous n'avons pas d'accident n'ayant que des indemnités: en effet, comme précisé dans le descriptif à notre disposition, ne sont recensés que les accidents impliquant au moins un véhicule et ayant fait au \*moins une victime ayant nécessité des soins\*).

A partir des 4 dataframes ci dessous, nous établirons donc le dataframe df\_total\_final:

- **Usagers** (1 ligne par usager, 619971 lignes)

	Num_Acc	id_usager	id_vehicule	num_veh	place	catu	grav	sexe	an_nais	trajet	secul	secu2	secu3	locp	actp	etatp
0	202200000001	1099 700	813 952	A01	1	1	3	1	2008.0	5	2	8	-1	-1	-1	
1	202200000001	1099 701	813 953	B01	1	1	1	1	1948.0	5	1	8	-1	-1	-1	
2	202200000002	1099 698	813 950	B01	1	1	4	1	1988.0	9	1	0	-1	0	0	
3	202200000002	1099 699	813 951	A01	1	1	1	1	1970.0	4	1	0	-1	0	0	
4	202200000003	1099 696	813 948	A01	1	1	1	1	2002.0	0	1	0	-1	-1	-1	
5	202200000003	1099 697	813 949	B01	1	1	4	2	1987.0	9	1	0	-1	-1	-1	
6	202200000004	1099 694	813 947	A01	1	1	1	2	2000.0	0	1	8	-1	-1	-1	
7	202200000004	1099 695	813 947	A01	10	3	4	2	1967.0	5	0	-1	-1	2	3	
8	202200000005	1099 692	813 945	B01	1	1	1	2	1975.0	0	1	0	-1	-1	-1	
9	202200000005	1099 693	813 946	A01	1	1	4	2	1996.0	0	1	0	-1	-1	-1	

Dans lequel nous avons effectué ces analyses et transformations:

#Travail sur la colonne "catu" pour obtenir une nouvelle colonne correspondant à chaque catégorie usager. Création df\_usage4

```

#Travail sur les 3 colonnes SECU, et décision de regrouper les 3 informations, en sommant les types de sécu des 3
colonnes lorsqu'ils sont identiques. Creation df_sécu

# Regroupement du DF_Secu par num acc et id véhicule pour obtenir 463183 lignes qui correspond au nombre de
véhicules total dans le data set, en vue de travailler ensuite le dataset véhicule. Création df_secu_final

#Merge de df_secu_final et df_usag4 précédent pour obtenir 63183 lignes. Crédation df_usag_5

#Travail sur la colonne "PLACE" en décision de regrouper les modalités en 5 catégories: conducteur, passager avant,
passager arrière, passager milieu et piéton. Crédation d'un df_usag_place à merger ensuite au df_usag5 précédent

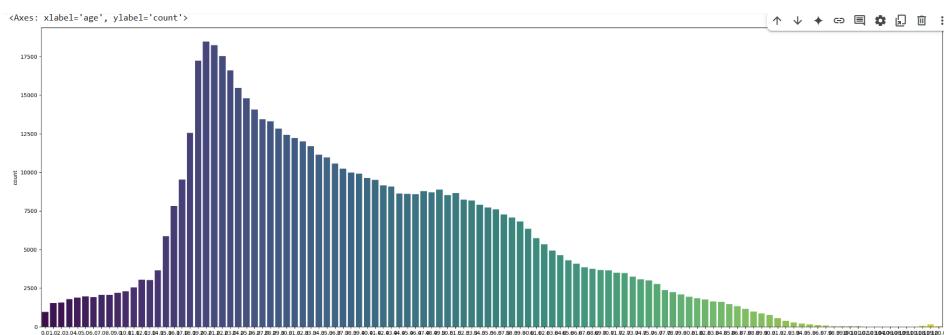
# L'info Conducteur est redondante puisque c'est une information que nous avions déjà grâce à la colonne CATU, de
même pour Piéton . Crédation df_usag_6

# Encodage de la colonne sexe afin d'obtenir 3 colonnes distinctes homme/femme /non renseigné. Crédation df_sex

#Ajout df_sex au df_usag6 et sauve mon df_usag7 qui est le df final pour la table usagers. Crédation df_usag7

#Ajout d'une variable age. Crédation df_age , puis étude de la distribution selon les ages pour établir des classes d'âge
pertinentes

```



```

# Etude des outliers de la variable âge (on constate sur le graphique qu'il y a un certain nombre d'accidents pour des
personnes de plus de 100ans, regardons en détail combien il y en a.

```

L'âge médian est de 35.0 ans

lower limit -20.5

upper limit 95.5

il y a 2060 outliers

```
# Catégorisation de l'âge en 4 tranches: 0-17, 18-60, 61-95, >96
```

```
#Création d'un df_age_final
```

```
# Merge de df_usag7 et df_age
```

```
## Enregistrement de df_usag8
```

- Véhicules** (qui compe une ligne par véhicule)

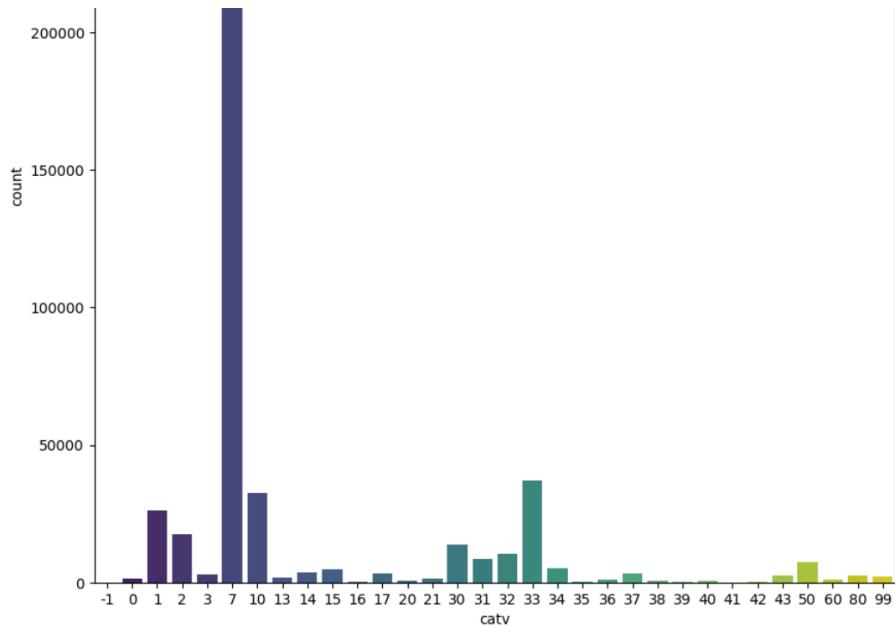
	Num_Acc	id_vehicule	num_veh	senc	catv	obs	obsm	choc	manv	motor	occutc	obstacle_fixe	obstacle_mobile
0	202100000001	201 764	B01	1	1	0	2	1	1	5	NaN	0	1
1	202100000001	201 765	A01	1	7	0	9	3	17	1	NaN	0	1
2	202100000002	201 762	A01	0	7	2	2	1	1	0	NaN	1	1
3	202100000002	201 763	B01	0	7	0	2	1	9	0	NaN	0	1
4	202100000003	201 761	A01	1	7	0	1	3	1	1	NaN	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
467164	201900058837	137 982 135	C01	1	7	0	2	1	2	1	NaN	0	1
467165	201900058838	137 982 132	A01	2	7	3	0	1	21	0	NaN	1	0
467166	201900058839	137 982 131	A01	2	33	0	0	7	1	1	NaN	0	0
467167	201900058840	137 982 129	B01	1	10	0	2	4	0	2	NaN	0	1
467168	201900058840	137 982 130	A01	1	10	0	2	1	0	1	NaN	0	1

467169 rows x 13 columns

Dans lequel nous avons effectué ces analyses et transformations:

#Pour les colonnes Obsm et Obs, on regarde la distribution des valeurs et on choisit de transformer ces 2 colonnes en binaire: 0 sans obs ; 1 si obstacle. Dans l'idée d'un modèle qui prédirait la gravité de l'accident grâce à un boîtier intégré dans le véhicule, on juge que le type d'obstacle percuté n'est pas une information que le boîtier pourrait connaître, en revanche il serait probablement capable de déterminer s'il y a eu un choc avec un obstacle ou non, d'où le choix de garder une variable binaire

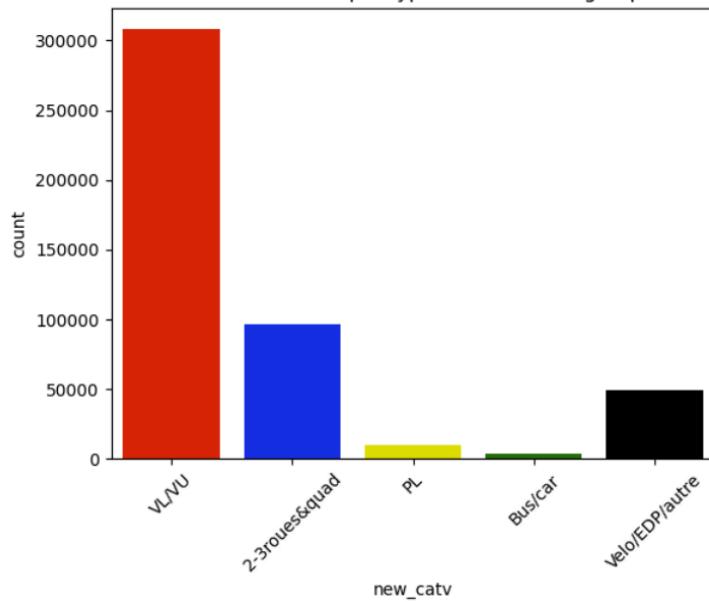
#Pour la colonne Catégorie Véhicule , nous regardons la distribution par type de véhicule avant de procéder à un regroupement: la catégorie 7 est largement représentée( Véhicule Léger ), viennent ensuite les 2 roues (moto, scooter et bicyclette).



Regroupement des catégories véhicules :

- 1 VL et V utilitaire
- 2 Moto, scooter, 2 roues, 3 roues, quad
- 3 Poids lourds
- 4 Bus/car
- 5 Velo/Vae/Edp et autres(trottinettes)

Distribution du nombre d'accident par type de véhicule regroupés en 5 catégories



```

#Pour la colonne CHOC , on choisit de regrouper de la manière suivante , après visualisation de la distribution
- 0 pas de choc
- 1-2-3 Avant
- 4-5-6 Arriere
- 7-8 coté
- 9 tonneaux

# On filtre le df en conservant les variables qui nous intéressent uniquement:
"Num_Acc","id_vehicule","new_catv","obstacle_fixe","obstacle_mobile","aucun_choc","choc_AV",
"choc_AR","choc_cote","choc_tonneaux"

# On transforme la colonne nouvelle_cat_véhicule(new_catv) en 1 colonne par type de véhicule et les renomme pour
une meilleure lisibilité. Création de df_veh1

# On rajoute une colonne qui comptabilise le nombre de véhicule

#On merge le df_veh1 avec df usagers final (df_usag8)

# On utilise le groupby par numéro d'accident en utilisant la fonction "sum" , par exemple pour les lignes d'index 0 et 1
du df merged précédent qui correspondent au même accident, on regroupe à 1 ligne par accident en sommant les
valeurs pour les autres colonnes (ex: usager_count devient 3): ainsi pour chaque colonne on connaît le nombre de
personnes concernées ( combien d'indemne, de tués, de passager, de piéton, de véhicule léger, de moto, etc, etc pour
chaque accident )

#On enregistre df_merged_usag_veh final

```

- Lieux

	Num_Acc	catr	voie	v1	v2	circ	nbv	vosp	prof	pr	pr1	plan	lartpc	larrout	surf	infra	situ	vma	
0	202000000001	4	HENRI BARBUSSE (AVENUE)	0.0	NaN	2	2	0	1	0	0	1	NaN	-1	1	0	1	50	
1	202000000002	4	MOUSSEAUX(CHEMIN)	0.0	NaN	2	2	0	1	0	100	3	NaN	-1	1	0	1	50	
2	202000000003	4	CARNOT(AVENUE)	0.0	NaN	-1	2	0	1	0	0	1	NaN	-1	1	0	1	50	
3	202000000004	4	VICTOR HUGO (AVENUE)	0.0	NaN	2	2	0	1	1	0	1	NaN	-1	1	0	1	30	
4	202000000005	3		35	0.0	NaN	1	1	0	1	4	674	2	NaN	-1	1	3	8	50
...	...	...		...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
289259	202200055298	3		71	-1.0	NaN	2	2	0	2	(1)	(1)	1	NaN	-1	1	0	8	80
289260	202200055299	3		973	-1.0	NaN	2	2	0	1	29	0	2	NaN	-1	1	0	3	80
289261	202200055300	3		22	0.0	D	2	2	0	1	39	553	2	NaN	-1	7	0	3	80
289262	202200055301	3		18	-1.0	D	2	2	0	1	30	125	1	NaN	-1	1	0	1	80
289263	202200055302	3		NaN	-1.0	NaN	3	4	0	1	(1)	(1)	1	NaN	4	1	0	1	70

289264 rows × 18 columns

Dans lequel nous avons effectué ces analyses et transformations:

```

# On constate dans un premier temps que le df_usag8 précédent, qui contient désormais 1 ligne par véhicule, compte
463184 lignes, alors que la table véhicule compte 467169 valeurs unique d'id_vehicules

# On crée 2 df pour me permettre de comparer et identifier les véhicules qui n'apparaissent pas dans la table usagers

#On filtre les colonnes qui nous intéressent, à savoir "Num_Acc","catr","circ","nbv","vma","surf"

#On établit une nouvelle variable catr: on choisit de regrouper les nationales/dep/communales d'un côté, les autoroute
de l'autre , les autres routes correspondent à – Hors réseau public – Parc de stationnement ouvert à la circulation
publique – Routes de métropole urbaine
- 1 Autoroute
- 2 nationale, départementale et communale 2,3,4
- 3 Autre 5,6,7,9

#On regroupe "Circ", le sens de circulation, en 2 modalités:
1 – A sens unique (initialement modalité 1)
2 – Bidirectionnelle (2 et autres)

#Surfaces ('surf'): au vu de la distribution, on regroupe comme suit :
- 1 normale
- 2 mouillée/enneigée

```

- 3 autre

Lorsqu'il y a 2 types de surfaces renseignés ( ce qui arrive notamment lorsque l'accident est survenu au niveau d'une intersection entre 2 routes, l'état de chacune des routes est parfois différent), on choisit de conserver l'état le plus accidentogène ( mouillé plutôt que normal)

# 272 valeurs non renseignées, on les garde ainsi en les incluant dans "autres"

count	
surf	count
1	232148
2	52688
9	1543
7	953
3	464
8	448
5	411
-1	272
6	219
4	118

#On fait ensuite un groupby par numéro d'accident , pour le nombre de voies(nbv), lorsqu'il y a 2 valeurs différentes ( dans le cas d'accidents survenue à une intersection ou un croisement par exemple), on conserve pour le moment le plus élevé, pour la VMA on choisit de conserver la plus élevée

# Sauvegarde le df\_lieux\_final dans le drive partagé , il contient désormais 273226 lignes , 1 ligne par accident

#### • Caractéristiques

	Num_Acc	jour	mois	an	hrmn	lum	dep	com	agg	int	atm	col	adr	lat	long
0	201900000001	30	11	2019	01:30	4	93	93053	1	1	1	2	AUTOROUTE A3	48,8962100	2,4701200
1	201900000002	30	11	2019	02:50	3	93	93066	1	1	1	6	AUTOROUTE A1	48,9307000	2,3688000
2	201900000003	28	11	2019	15:15	1	92	92036	1	1	1	4	AUTOROUTE A86	48,9358718	2,3191744
3	201900000004	30	11	2019	20:20	5	94	94069	1	1	1	4	A4	48,8173295	2,4281502
4	201900000005	30	11	2019	04:00	3	94	94028	1	1	1	2	A86 INT	48,7763620	2,4332540
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
273221	202300054818	26	10	2023	20:45	5	974	97416	2	1	1	6	LA FONTAINE (RUE JEAN DE)	-21,33828000	55,47771000
273222	202300054819	26	10	2023	19:10	3	974	97416	1	1	1	3	RN3 (ANCIENNE ROUTE)	-21,28865000	55,50994000
273223	202300054820	26	10	2023	09:40	1	974	97411	2	1	1	7	BAMBOU (CHEMIN DE LA RUELLE)	-20,90129000	55,40598000
273224	202300054821	26	10	2023	17:20	1	973	97302	1	6	1	3	ROUTE NATIONALE 1	4,89713000	-52,32854000
273225	202300054822	20	10	2023	16:30	1	69	69387	2	1	6	3	Boulevard Yves Farge	45,73306000	4,82540000

273226 rows × 15 columns

Dans lequel nous avons effectué ces analyses et transformations:

# Merge de la table lieux précédente avec la table caractéristiques

# Importation de la table précédemment mergée entre usagers et véhicules, puis merge des 2 tables afin d'obtenir le df\_final complet et agrégé à 1 ligne apr accident

# On renomme quelques colonnes pour une meilleure compréhension:

{"new\_catv\_1":"VL\_VU","new\_catv\_2":"2roues\_3roues","new\_catv\_3":"PL","new\_catv\_4":"bus\_car","new\_catv\_5":"velo\_trc

# Ajout d'une colonne qui détermine la gravité de l'accident, nous décidons de conserver la gravité la plus importante lorsque plusieurs usagers ont des gravités différentes ( par exemple si un accident compte un blessé léger et un tué, nous considérons cet accident comme mortel ):

- 4 - accident mortel
- 3 - accident avec blessés hospitalisés
- 2 - accident avec blessés légers
- 1 - accident avec des indemnités

```
# On supprime les colonnes inutiles pour notre étude (adr, com id_vehicule, lum)
# On supprime les colonnes redondantes (atu_conducteur, passager et catu_pieton sont redondantes avec la place de l'usager)
# On supprime les colonnes non renseignées, qui n'apportent aucune information (sexe non renseigné, total_secu_non_renseigné, total_secu-non_déterminé, grav_non_renseignée)
# On exclut les outliers de la variable age (>96)
# On enregistre le df_total_final dans le drive partagé
```

Nous obtenons un dataframe df\_total\_final dont voici les premières lignes et les infos

	Num_Acc	jour	mois	an	hrmn	dep	agg	int	atm	col	...	choc_AR	choc_cote	choc_tonneaux	VL_VU	2roues_3roues_quad	PL	bus_car	velo_trott_edp	nbr_veh	gravité_accident
0	201900000001	30	11	2019	01:30	93	1	1	1	2	...	1	0	0	1	0	0	0	1	2	2
1	201900000002	30	11	2019	02:50	93	1	1	1	6	...	0	0	0	1	0	0	0	0	1	2
2	201900000003	28	11	2019	15:15	92	1	1	1	4	...	2	0	0	3	0	0	0	0	3	2
3	201900000004	30	11	2019	20:20	94	1	1	1	4	...	2	0	0	3	0	0	0	0	3	2
4	201900000005	30	11	2019	04:00	94	1	1	1	2	...	2	0	0	2	0	0	0	0	2	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
273221	202300054818	26	10	2023	20:45	974	2	1	1	6	...	0	1	0	1	0	0	0	0	1	2
273222	202300054819	26	10	2023	19:10	974	1	1	1	3	...	0	1	0	1	1	0	0	0	2	3
273223	202300054820	26	10	2023	09:40	974	2	1	1	7	...	1	0	0	1	0	0	0	0	1	2
273224	202300054821	26	10	2023	17:20	973	1	6	1	3	...	0	0	0	1	1	0	0	0	2	2
273225	202300054822	20	10	2023	16:30	69	2	1	6	3	...	0	1	0	1	0	0	0	1	2	3

273226 rows × 60 columns

Mounted at /content/drive

```
<ipython-input-3-3fb84b0cf0b3>:9: 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/Projet_accidents/Dataset/2019_2023/df_total_final.csv"
```

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

RangelIndex: 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_mouillée_enneigée	273226	non-null int64

```

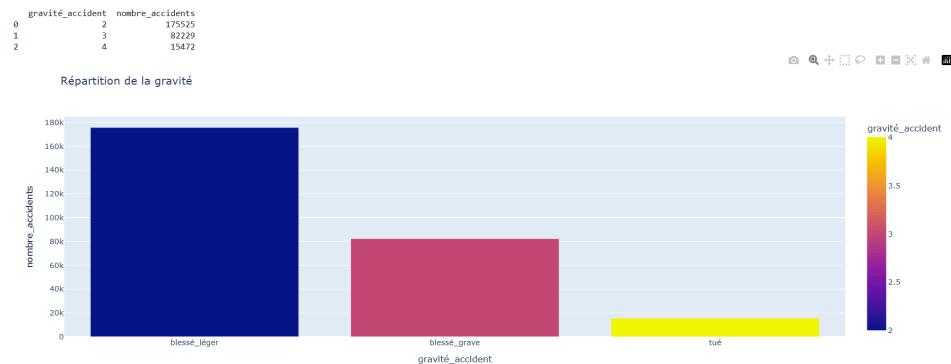
21 etat_route_autre      273226 non-null int64
22 usager_count          273226 non-null int64
23 indemne               273226 non-null int64
24 tué                   273226 non-null int64
25 blessé_hospitalisé   273226 non-null int64
26 blessé_léger          273226 non-null int64
27 total_sans_secu       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

```

## 4. Analyse des données

Nous importons les bibliothèques nécessaires, et transformons la date au format datetime.

Nous commençons par afficher la distribution de la gravité des accidents.



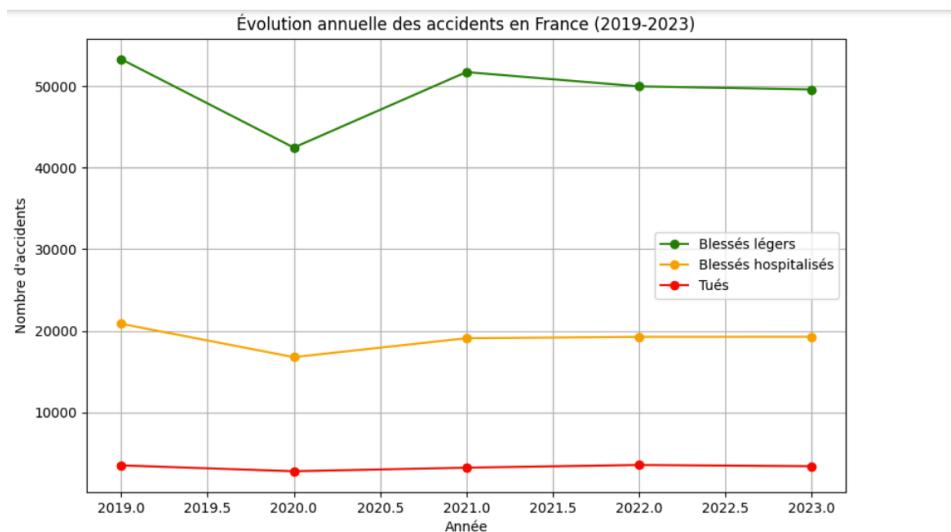
Nous constatons un déséquilibre de la répartition des valeurs de la variable cible (gravité de l'accident), qu'il faudra garder en tête pour nos modèles de machine learning. Nous constatons qu'il n'y a aucun accident avec la gravité 'indemne', ce qui signifie que pour chaque accident répertorié, il y a au moins un blessé.

### a. Conditions de l'accident

- Date

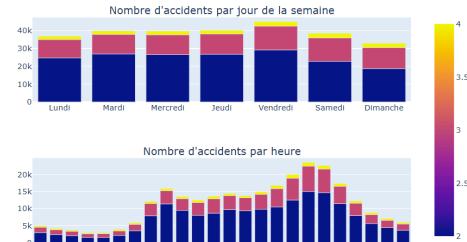
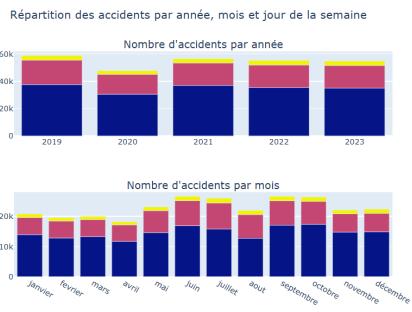
La première question que nous nous posons est la suivante: existe-t-il une corrélation entre le moment de l'accident et sa gravité?

Analysons en premier lieu l'évolution annuelle des accidents en France par catégorie de gravité

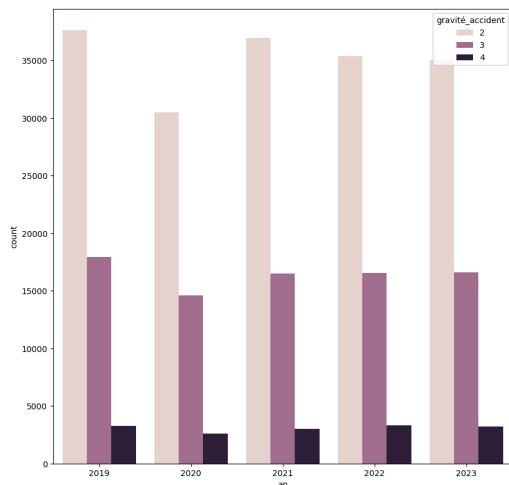


La répartition blessés légers, blessés hospitalisés et tués est globalement constante dans le temps (sauf en 2019, année covid). Mais nous avons pu constater des variations selon les mois, et selon les jours de la semaine. Nous gardons cela en tête pour éventuellement analyser la série temporelle et prédire les nombres de blessés légers, hospitalisés, et graves en fonction des données du passé, par département par exemple. Cela permet d'ajuster en avance les moyens humains et matériels. Nous effectuerons ce travail dans la deuxième partie (prédition de la gravité).

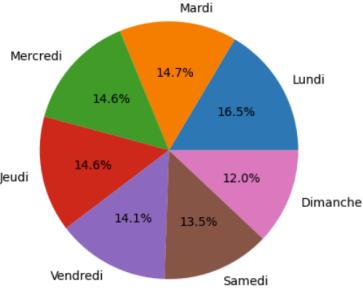
Nous nous intéressons maintenant à la répartition du nombre d'accidents par année, jour de la semaine, mois et heure de la journée.



### Répartition de la gravité des accidents par année



### Répartition des accidents par jour de la semaine



Le nombre d'accidents par an est à peu près stable, sauf l'année 2020 qui est l'année du covid. Il y a plus d'accidents en juin, juillet, septembre, octobre, et le vendredi, et entre 17 et 18h.

Enfin, nous réalisons un test statistique pour confirmer une corrélation entre l'heure de l'accident et la gravité de l'accident. Le test de chi2 confirme que l'heure influence significativement la gravité de l'accident.

```
from scipy.stats import chi2_contingency
contingence = pd.crosstab(df['date'].dt.hour, df['gravité_accident'])

# Appliquer le test du Chi-carré
chi2, p_value, dof, expected = chi2_contingency(contingence)

# Interprétation
if p_value < 0.05:
    print("L'heure influence significativement la gravité de l'accident (p < 0.05).")
else:
    print("Aucune influence significative de l'heure sur la gravité de l'accident.)
```

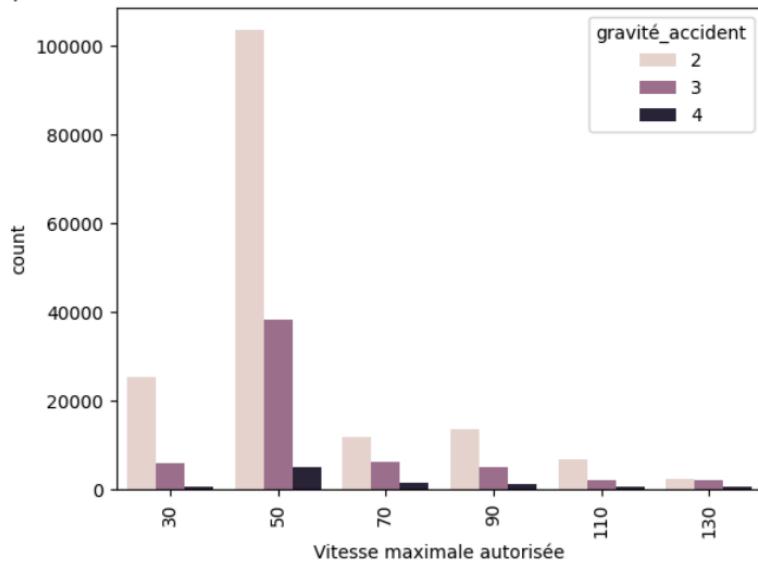
L'heure influence significativement la gravité de l'accident ( $p < 0.05$ ).

- **Vitesse maximale autorisée**

Existe t-il un lien entre la vitesse maximale autorisée et la gravité de l'accident?

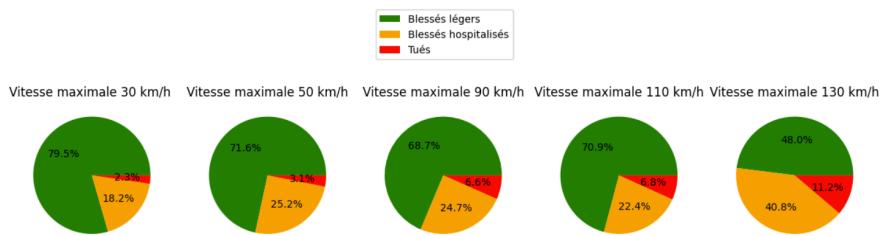
Nous supprimons les vitesses qui sont mal renseignées (0, 500...) pour analyser la répartition des accidents selon la vitesse maximale autorisée et la gravité.

Répartition du nombre d'accidents selon la vitesse maximale autorisée et la gravité



Etudions à présent la répartition de la gravité des accidents pour les différentes vitesses maximales autorisées.

Répartition des accidents par gravité pour différentes vitesses

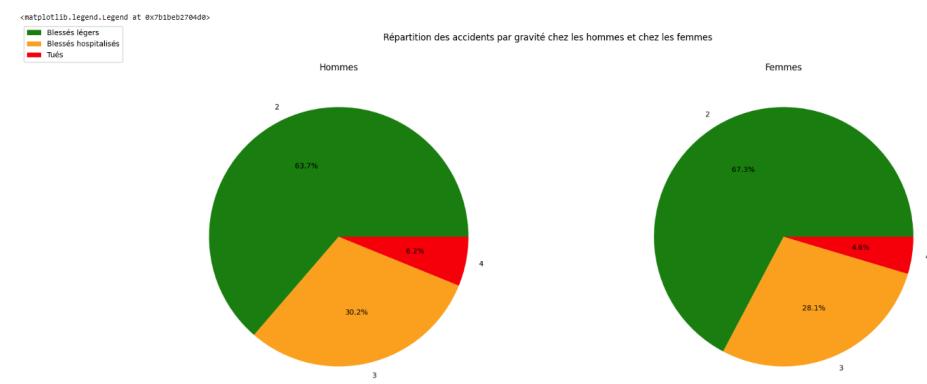
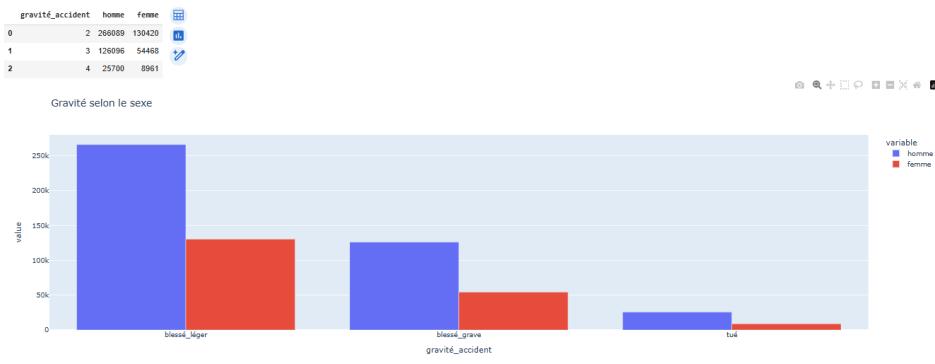


Il y a beaucoup plus d'accidents sur les routes limitées à 50 km/h. Les accidents les plus graves sont sur les routes à 130 km/h.

## b. Usagers

Nous souhaiterions savoir s'il existe un lien entre le sexe et la gravité de l'accident. Pour cela, analysons la distribution de la gravité des accidents selon la variable sexe

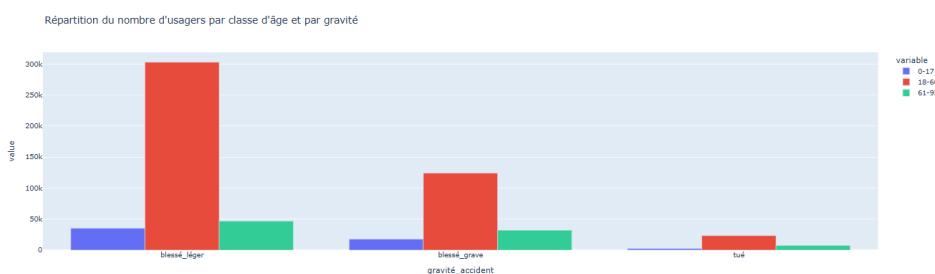
- **Variable sexe**

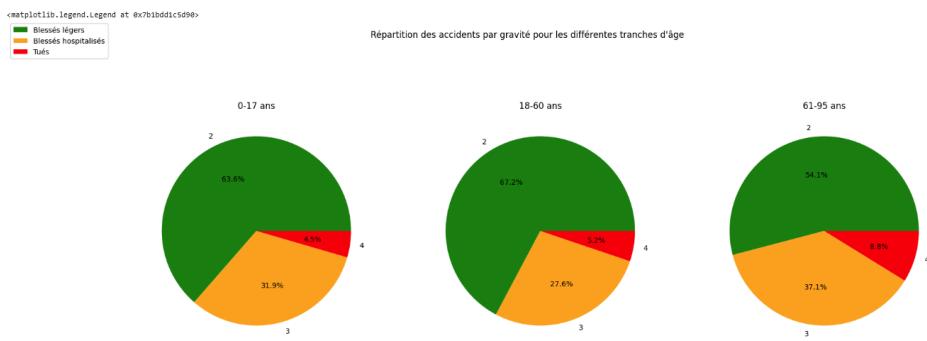


Les hommes ont plus d'accidents que les femmes, et la proportion d'accidents mortels ou graves chez les hommes est aussi plus élevée que chez les femmes. La variable sexe semble influencer la gravité de l'accident.

#### • Variable Age

Il est également intéressant d'analyser un éventuel lien entre l'âge des victimes et la gravité de l'accident.

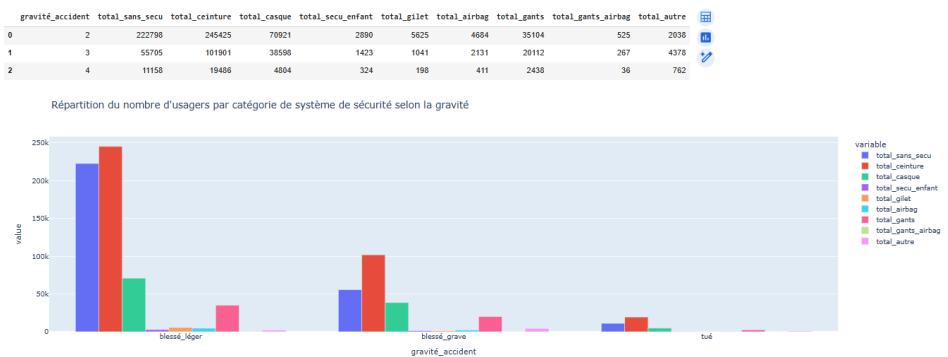




Il y a évidemment beaucoup plus de 18\_60 ans impliqués dans les accidents puisque la tranche d'âge est plus large. On constate que la proportion d'accidents graves (tués et blessés hospitalisés) est plus élevée chez les 61-96 ans. L'âge semble donc impacter la gravité de l'accident.

- **Variable système de sécurité**

Nous nous interrogeons également sur un lien éventuel entre le système de sécurité et la gravité de l'accident.



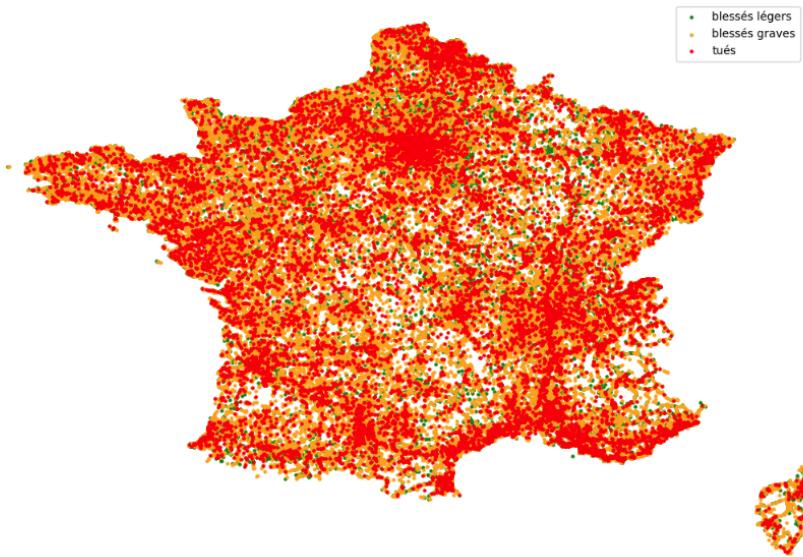
On constate que l'équipement de sécurité majoritaire est le port de ceinture, et que l'absence de ceinture arrive en seconde position, quelle que soit la gravité de l'accident.

### c. Localisation de l'accident

Pour analyser des tendances selon la localisation, il nous paraît pertinent de réaliser une carte géographique des accidents.

- **Carte de la répartition géographique des accidents**

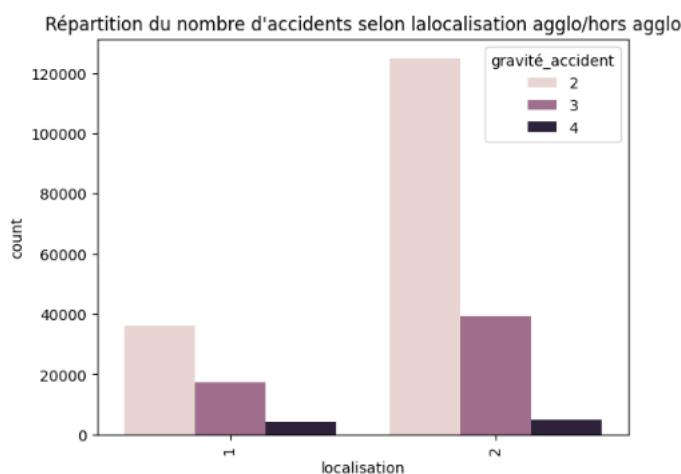
Représentation de la gravité des accidents en France entre 2019 et 2023



L'Île de France regroupe le plus d'accidents mortels. La région PACA et Lyon ont également beaucoup de blessés graves et du tués. On constate une plus forte concentration d'accidents dans les villes.

- **Localisation agglomération/hors agglomération**

Précisons cette répartition géographique, en étudiant la variable 'agg', pour laquelle la valeur 1 signifie que l'accident a eu lieu hors agglomération, et 2 en agglomération.



Il y a beaucoup plus d'accidents en agglomération qu'hors agglomération, ce que l'on avait déjà noté sur la carte géographique. Les accidents sont plus graves en dehors des agglomérations. On peut faire l'hypothèse que la localisation(agglo/hors aggo) influence la gravité de l'accident. Vérifions par un test statistique.

```
contingence_agg = pd.crosstab(df['gravité_accident'], df['agg'], normalize = True)
contingence_agg = pd.DataFrame(contingence_agg).apply(lambda x: round(x, 4)*100, axis = 1)
print(contingence_agg)

# Créer la heatmap avec Plotly Express
sns.heatmap(contingence_agg, annot=True, cmap='Blues')
```

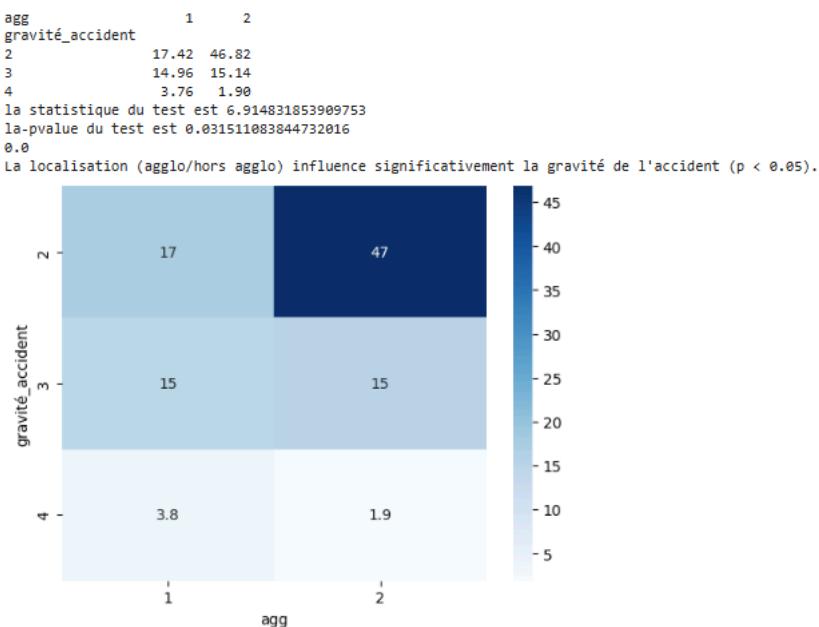
```

results = chi2_contingency(contingence_agg)
print ('la statistique du test est', results[0])
print ('la-pvalue du test est', results[1])
# Appliquer le test du Chi-carré

print(p_value)
# Interprétation:

if p_value < 0.05:
    print("La localisation (agglo/hors agglo) influence significativement la gravité de l'accident (p < 0.05).")
else:
    print("Aucune influence significative de la localisation (agglo/hors agglo) sur la gravité de l'accident.")

```



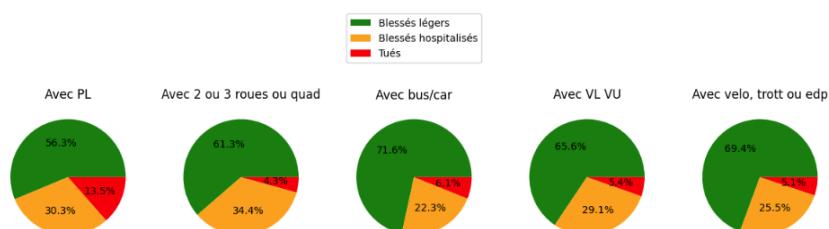
Le test confirme cette corrélation entre la localisation et la gravité de l'accident.

## d. Véhicules

- Types de véhicules impliqués**

Nous pouvons supposer que le type de véhicule impliqué influe sur la gravité de l'accident.

Etudions la tendance de cette relation.

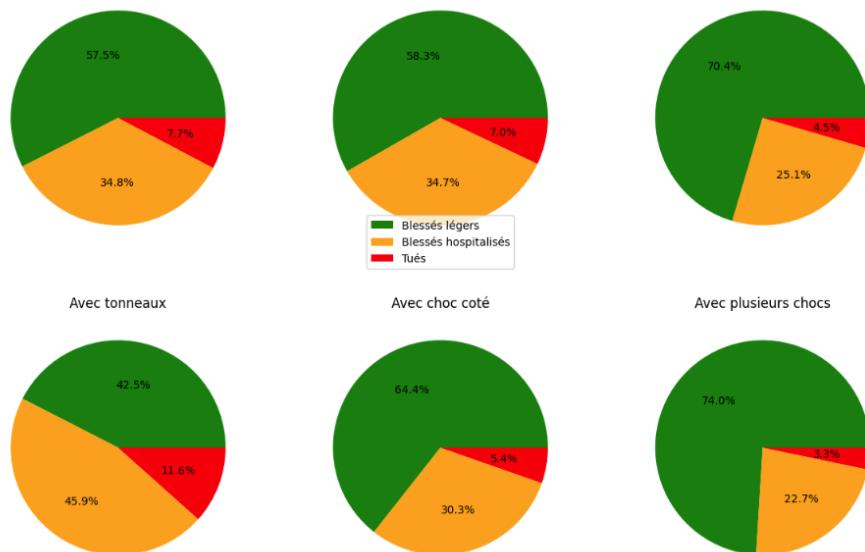


Il semble en effet que les accidents les plus mortels soient ceux impliquant au moins un poids lourd.

- **Point de choc**

Etudions la relation entre la variable 'point de choc' et la gravité de l'accident.

Répartition des accidents par gravité pour différents types de chocs

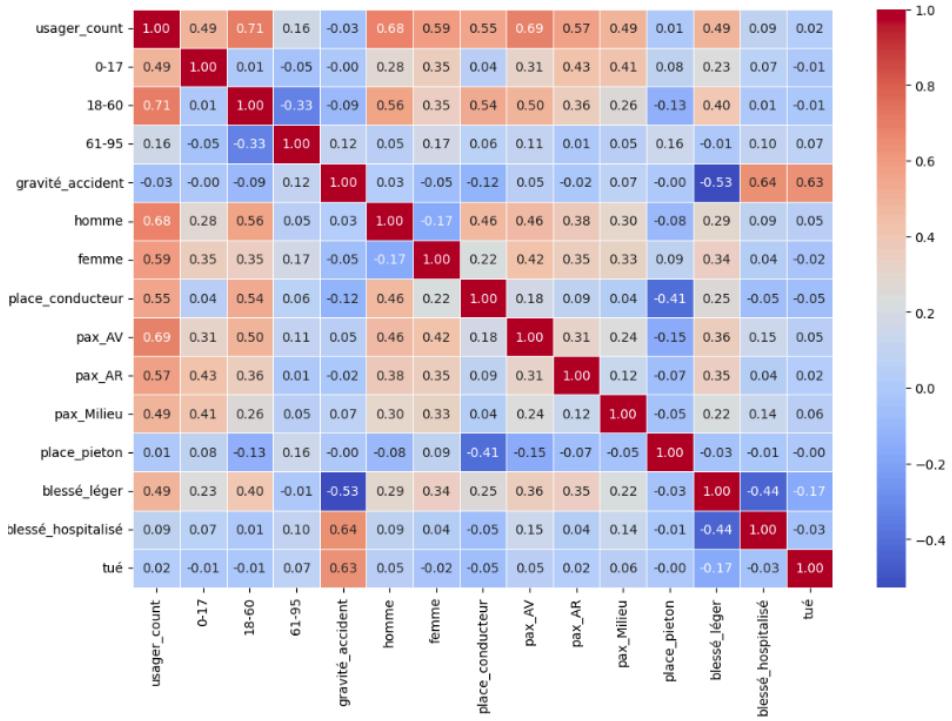


Les graphiques donnent à penser que les accidents les plus graves sont ceux pour lesquel il y a eu au moins un choc tonneaux, et qu'ils sont également les plus mortels.

La dernière étape de notre analyse consiste à établir des matrices de corrélation entre différentes variables.

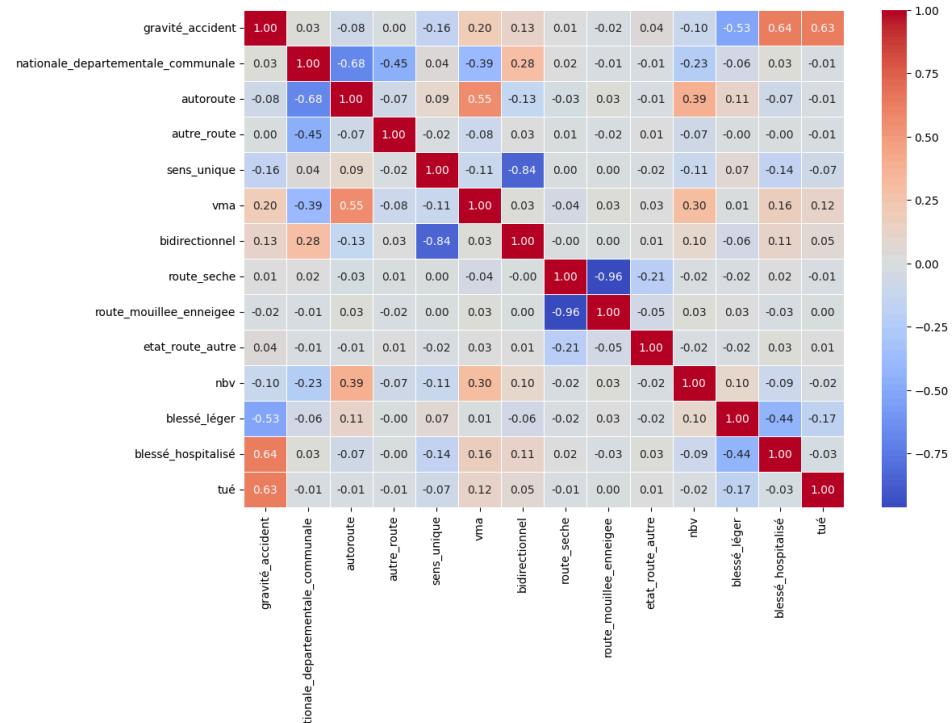
## e. Corrélation usagers

- **Corrélation selon les caractéristiques des usagers**



On observe une corrélation positive entre la tranche d'âge 61\_95 ans et la gravité de l'accident. De même, on observe une corrélation positive entre le nombre d'hommes impliqués et la gravité de l'accident. Cela corrobore nos hypothèses précédentes.

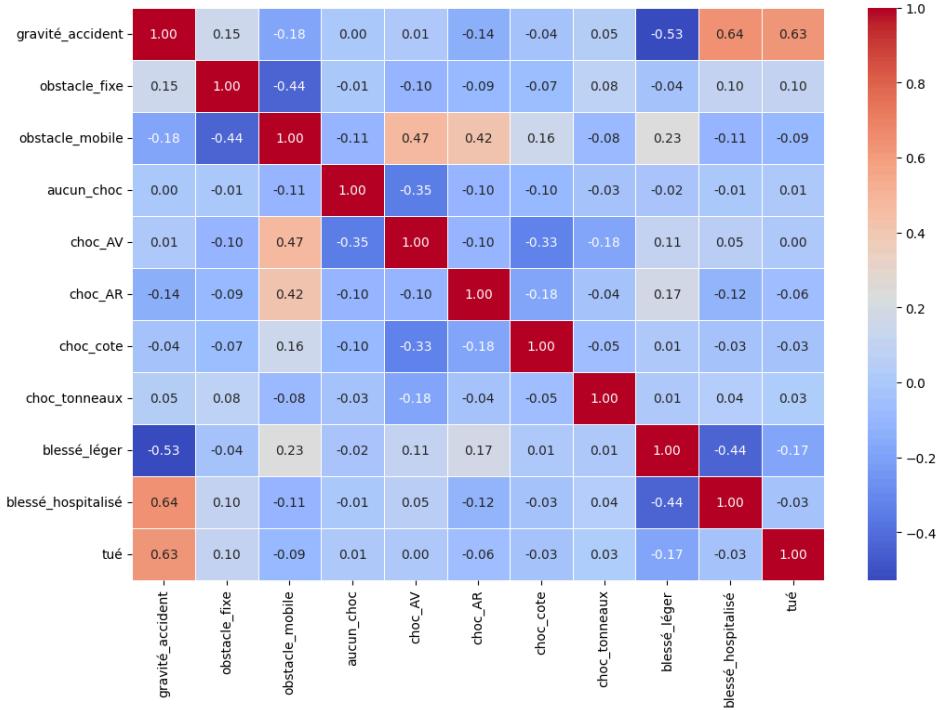
#### • Corrélations selon les conditions de routes



On observe une corrélation positive entre la vitesse maximale autorisée et la gravité de l'accident: plus la vitesse maximale autorisée augmente, plus la gravité de l'accident augmente.

On observe également une corrélation positive forte entre la bidirectionnalité et la gravité.

- **Corrélation selon les obstacles et types de chocs**



On observe une corrélation positive entre le nombre de véhicules ayant heurté un obstacle fixe et la gravité de l'accident. Et également une corrélation positive, mais moins forte, entre le nombre de véhicules ayant fait un choc\_tonneaux et la gravité de l'accident.

Nous avons mis en évidence des corrélations entre plusieurs variables de notre dataframe et la gravité de l'accident. Nous essaierons donc, dans la prochaine étape de modélisation, de conserver un maximum ces variables.

## II. Modélisation pour prédire la gravité d'un accident

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 possibles.

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 gravités 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 ressais 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:

- Feature engeneering pour 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\_scoring 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.

## 1. Feature engeneering

#Conservation de l'heure de l'accident et suppression le reste de la date.

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

#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. On obtient la répartition suivante:

```

    count
nbv
  2    178149
  4    31627
  1    27445
  3    21215
  6     6864
  5    4219
  8    2125
  7     755
 10     510
  9     317
dtype: int64

```

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

#Enrgistrement du df\_machine\_learning

Comme indiqué précédemment, nous avons testé de travailler sur une série temporelle permettant de prédire le nombre de blessés hospitalisés, mais il n'a pas été concluant. Les codes correspondants seront cependant présentés à la fin de ce document. Les codes associés à tous les essais de machine learning seront également présentés à la fin du document. Ici, nous ne présenterons que les résultats de ces tests.

## 2. Meilleurs résultats avec modèles 3 classes

### a. Modèle 3 classes Random Forest

```

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

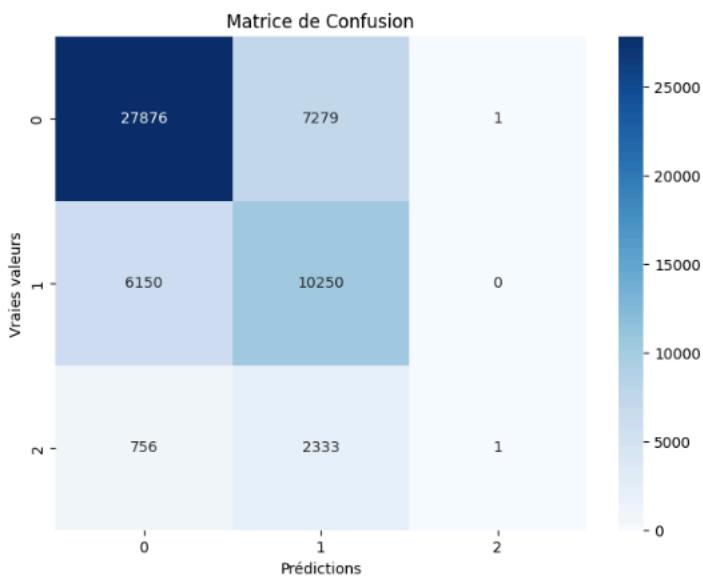
Meilleure configuration de pondération : {0: 1, 1: 2, 2: 1}

**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
      macro avg       0.61      0.47      0.45     54646
      weighted avg    0.70      0.70      0.68     54646

```



### b. Modèle 3 classes Logistic Regression

```

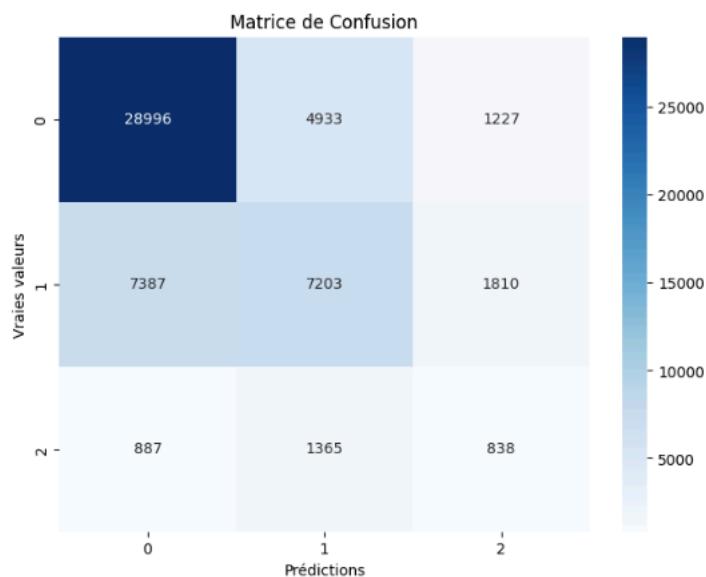
Meilleurs paramètres :
{'classifier__C': 0.01, 'classifier__solver': 'liblinear'}

**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
macro avg       0.51     0.51     0.51    54646
weighted avg    0.67     0.68     0.67    54646

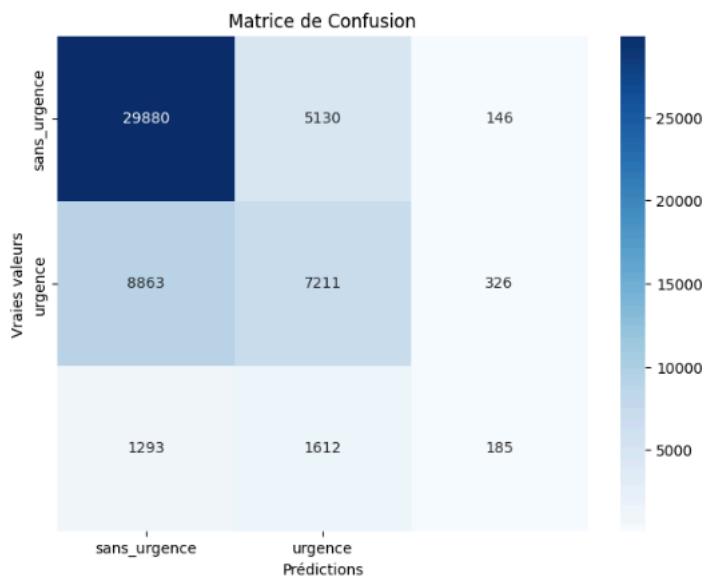
```



### c. Modèle 3 classes KNN

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

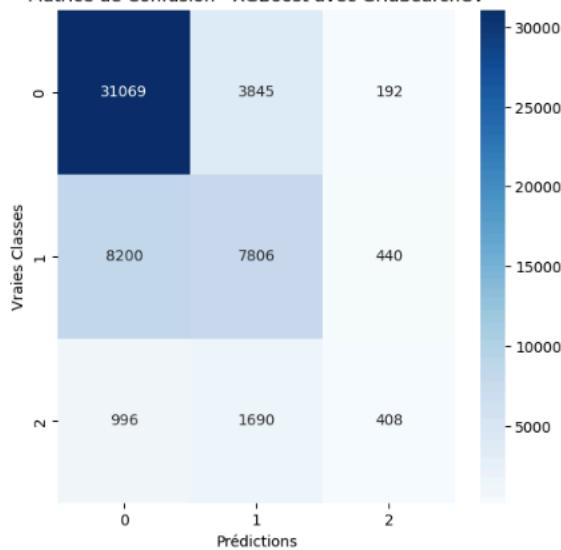
Rapport de Classification :					
	precision	recall	f1-score	support	
0	0.75	0.85	0.79	35156	
1	0.52	0.44	0.48	16400	
2	0.28	0.06	0.10	3890	
accuracy			0.68	54646	
macro avg	0.51	0.45	0.46	54646	
weighted avg	0.65	0.68	0.66	54646	



#### d. Modèle 3 classes XGBoost

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

### 3. Meilleurs résultats avec modèles 2 classes

#### a. Modèle 2 classes Random Forest

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

#### b. Modèle 2 classes Logistic Regression

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

#### c. Modèle 2 classes KNN

```

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
macro avg       0.63     0.62     0.62    54646
weighted avg    0.68     0.70     0.69    54646

```

#### d. Modèle 2 classes XGBoost

```

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
macro avg       0.67     0.70     0.68    54646
weighted avg    0.74     0.70     0.71    54646

```

## 4. 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	Hospitalis	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': 'lbfgs'}]	0.60	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	Hospitalis	Tué
Simple											
Oversampling											
Undersampling											
Grid search	{'classifier__n_neig': 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	Hospitalis	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.

Contrairement à nos attentes, la performance des modèles n'est pas significativement améliorée en réduisant notre variable cible à deux classes.

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.

## III. Codes

### 1. Nettoyage et analyse des données

Vous trouverez ci-dessous tous les codes utilisés pour élaborer les dataframes et les analyser

```
#####
# Montage du drive #####
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 permet d'avoir un fichier mis à jour si modifi

#####
# Création all_data_vehicules (2019-2023)
# Chemin vers le dossier contenant les fichiers CSV
folder_path = '/content/drive/My Drive/Datascientest/Dataset_perso/Vehicules/Vehicules_par_annee'
```

```

# Lister tous les fichiers dans le dossier
all_files = os.listdir(folder_path)

# Liste pour stocker les DataFrames
dataframes_vehicules = []

# Itérer à travers chaque fichier
for file in all_files:
    file_path = os.path.join(folder_path, file)

    if file.endswith('.csv'):
        # Lire le fichier comme texte
        with open(file_path, 'r') as f:
            content = f.read()

        # Remplacer plusieurs types de séparateurs par un séparateur unique (par exemple, une tabulation)
        content = content.replace(';', '\t').replace(';', '\t').replace('|', '\t') # Ajoutez d'autres séparateurs si nécessaire

        # Utiliser pd.read_csv avec le séparateur commun
        df_vehicules = pd.read_csv(StringIO(content), sep='\t', low_memory=False) # permet de mieux gérer les données

        # Ajouter le DataFrame à la liste
        dataframes_vehicules.append(df_vehicules)

# Concaténer tous les DataFrames
all_data_vehicules_2019_2023 = pd.concat(dataframes_vehicules, ignore_index=True)
#enregistrer dans le fichier concaténé dans le drive

all_data_vehicules_2019_2023.to_csv('/content/drive/My Drive/Datascientest/Dataset_perso/all_data_vehicules_2019_2023.csv')

##### Création du fichier all_data_usagers_2019_2023 #####
# Chemin vers le dossier contenant les fichiers CSV
folder_path = '/content/drive/My Drive/Datascientest/Dataset_perso/Usagers'

# Lister tous les fichiers dans le dossier
all_files = os.listdir(folder_path)

# Liste pour stocker les DataFrames
dataframes_usagers = []

# Itérer à travers chaque fichier
for file in all_files:
    file_path = os.path.join(folder_path, file)

    if file.endswith('.csv'):
        # Lire le fichier comme texte
        with open(file_path, 'r') as f:
            content = f.read()

        # Remplacer plusieurs types de séparateurs par un séparateur unique (par exemple, une tabulation)
        content = content.replace(';', '\t').replace(';', '\t').replace('|', '\t') # Ajoutez d'autres séparateurs si nécessaire

        # Utiliser pd.read_csv avec le séparateur commun

```

```

df_usagers = pd.read_csv(StringIO(content), sep='\t', low_memory=False) # permet de mieux gérer les données

# Ajouter le DataFrame à la liste
dataframes_usagers.append(df_usagers)

# Concaténer tous les DataFrames
all_data_usagers_2019_2023 = pd.concat(dataframes_usagers, ignore_index=True)

all_data_usagers_2019_2023.to_csv('/content/drive/My Drive/Datascientest/Dataset_perso/all_data_usagers_2019_2023.csv')

#####
# Création all_data_carac_2019_2023 #####
# Chemin vers le dossier contenant les fichiers CSV
folder_path = '/content/drive/My Drive/Datascientest/Dataset_perso/Caractéristiques'

# Lister tous les fichiers dans le dossier
all_files = os.listdir(folder_path)

# Liste pour stocker les DataFrames
dataframes_carac = []

# Itérer à travers chaque fichier
for file in all_files:
    file_path = os.path.join(folder_path, file)

    if file.endswith('.csv'):
        # Lire le fichier comme texte
        with open(file_path, 'r') as f:
            content = f.read()

        # Remplacer plusieurs types de séparateurs par un séparateur unique (par exemple, une tabulation)
        content = content.replace(';', '\t').replace(';', '\t').replace('|', '\t') # Ajoutez d'autres séparateurs si nécessaire

        # Utiliser pd.read_csv avec le séparateur commun
        df_carac = pd.read_csv(StringIO(content), sep='\t', low_memory=False) # permet de mieux gérer les données de

# Ajouter le DataFrame à la liste
dataframes_carac.append(df_carac)

# Concaténer tous les DataFrames
all_data_carac_2019_2023 = pd.concat(dataframes_carac, ignore_index=True)

all_data_carac_2019_2023.to_csv('/content/drive/My Drive/Datascientest/Dataset_perso/all_data_carac_2019_2023.csv')

#####
# Création all_data_lieux_2019_2023 #####
# Chemin vers le dossier contenant les fichiers CSV
folder_path = '/content/drive/My Drive/Datascientest/Dataset_perso/Lieux'

# Lister tous les fichiers dans le dossier
all_files = os.listdir(folder_path)

# Liste pour stocker les DataFrames
dataframes_lieux = []

# Itérer à travers chaque fichier
for file in all_files:

```

```

file_path = os.path.join(folder_path, file)

if file.endswith('.csv'):
    # Lire le fichier comme texte
    with open(file_path, 'r') as f:
        content = f.read()

    # Remplacer plusieurs types de séparateurs par un séparateur unique (par exemple, une tabulation)
    content = content.replace(';', '\t').replace(';', '\t').replace('|', '\t') # Ajoutez d'autres séparateurs si nécessaire

    # Utiliser pd.read_csv avec le séparateur commun
    df_lieux = pd.read_csv(StringIO(content), sep='\t', low_memory=False) # permet de mieux gérer les données de c

    # Ajouter le DataFrame à la liste
    dataframes_lieux.append(df_lieux)

# Concaténer tous les DataFrames
all_data_lieux_2019_2023 = pd.concat(dataframes_lieux, ignore_index=True)
all_data_lieux_2019_2023.to_csv('/content/drive/My Drive/Datascientest/Dataset_perso/all_data_lieux_2019_2023.csv')

```

```

#####
##### Travail table véhicules #####
#####

## Importation de la table Véhicules - 1 ligne par véhicule impliqué
import pandas as pd
df_veh=pd.read_csv("/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/all_data_vehicules_2
df_veh.info() # 467169 lignes

## Pour les colonnes Obsm et Obs, on regarde la distribution des valeurs
print(df_veh["obsm"].value_counts()) # la modalité 2 signifie qu'un autre véhicule a été heurté, 0 signifie aucun obstacle
print(df_veh["obs"].value_counts()) # 0 signifie aucun obstacle fixe heurté, 1 signifie un autre véhicule en stationnement

# On choisit de transformer ces 2 colonnes en binaire: 0 sans obs ; 1 si obstacle. Dans l'idée d'un modèle qui prédirait
df_veh["obstacle_fixe"]=0
df_veh["obstacle_fixe"] = df_veh["obs"].apply(lambda x: 0 if x == 0 else 1)

df_veh
df_veh["obstacle_mobile"]=0
df_veh["obstacle_mobile"] = df_veh["obsm"].apply(lambda x: 0 if x == 0 else 1)

df_veh

#####
##### Catégorie Véhicule #####
#####

# Nous regardons la distribution par type de véhicule avant de procéder à un regroupement:
# La catégorie 7 est largement représentée( Véhicule Léger ), viennent ensuite les 2 roues (moto, scooter et bicyclette)

#Regroupement des catégories véhicules :
#1 VL et V utilitaire
#2 Moto, scooter, 2 roues, 3 roues, quad
#3 Poids lourds
#4 Bus/car
#5 Velo/Vae/Edp et autres(trottinettes)

df_veh["catv"].value_counts()
import matplotlib.pyplot as plt

```

```

import seaborn as sns

plt.figure(figsize=(10,10))
sns.countplot(data=df_veh, x='catv', palette="viridis")

def replace_catv(catv):
    if catv in [7,10]:
        return 1
    if catv in [2,30,31,32,33,34,35,36,41,42,43]:
        return 2
    if catv in [13,14,15]:
        return 3
    if catv in [37,38]:
        return 4
    else:
        return 5

df_veh['new_catv'] = df_veh['catv'].apply(replace_catv)

df_veh=df_veh.drop("catv",axis=1)

palette=["red","blue","yellow","green","black","purple"]
sns.countplot(data=df_veh, x='new_catv', palette=palette)

plt.xticks(ticks=[0,1, 2, 3, 4], labels=["VL/VU", "2-3roues&quad", "PL", "Bus/car", "Velo/EDP/autre"], rotation=45);
plt.title("Distribution du nombre d'accident par type de véhicule regroupés en 5 catégories");

plt.show()

#Pour la colonne CHOC , on choisit de regrouper de la manière suivante , après visualisation de la distribution
#0 pas de choc
# 1-2-3 Avant
#4-5-6 Arrière
#7-8 coté
#9 tonneaux

df_veh["choc"].value_counts()

df_veh["aucun_choc"]=0
df_veh["choc_AV"]=0
df_veh["choc_AR"]=0
df_veh["choc_cote"]=0
df_veh["choc_tonneaux"]=0
def replace_choc(row):
    if row["choc"] in [1,2,3]:
        row["choc_AV"]+=1
    if row["choc"] in [4,5,6]:
        row["choc_AR"]+=1
    if row["choc"] in [7,8]:
        row["choc_cote"]+=1
    if row["choc"] in [9]:
        row["choc_tonneaux"]+=1
    if row["choc"] in [0]:

```

```

row["aucun_choc"]+=1
return row

df_veh = df_veh.apply(replace_choc, axis=1)

# On filtre le df en conservant les variables qui nous intéressent uniquement
df_veh_filtre=df_veh[["Num_Acc","id_vehicule","new_catv","obstacle_fixe","obstacle_mobile","aucun_choc","choc_AV","choc_usag","choc_usag_fixe","choc_usag_mobile","choc_usag_fixe_fixe","choc_usag_fixe_mobile","choc_usag_mobile_fixe","choc_usag_mobile_mobile"]]

# On transforme la colonne nouvelle_cat_véhicule(new_catv) en 1 colonne par type de véhicule et les renomme pour la suite
df_veh1=pd.get_dummies(df_veh_filtre,columns=["new_catv"],dtype="int")
df_veh1=df_veh1.rename({"new_catv_1":"VL_VU","new_catv_2":"2roues_3roues_quad","new_catv_3":"PL","new_catv_4":"MOT"})


# On rajoute une colonne qui comptabilise le nombre de véhicule
df_veh1["nbr_veh"]=0
nbr_veh_counts = df_veh1['id_vehicule'].value_counts().to_dict()
df_veh1['nbr_veh'] = df_veh1['id_vehicule'].map(nbr_veh_counts)
df_veh1

#On merge le df_veh1 avec df usagers final (df_usag8)

df_merged_usag_veh=df_usag8.merge(df_veh1,on=["Num_Acc","id_vehicule"],how="left")

df_merged_usag_veh

# On utilise le groupby par numéro d'accident en utilisant la fonction "sum" , par exemple pour les lignes d'index 0 et 1
# ainsi pour chaque colonne on connaît le nombre de personnes concernées ( combien d'indemne, de tués, de passagers )
df_merged_usag_veh_final=df_merged_usag_veh.groupby("Num_Acc").agg("sum").reset_index()
df_merged_usag_veh_final.drop("id_vehicule",axis=1)

#On enregistre df_merged_usag_veh final
df_merged_usag_veh_final.to_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_merged_usag_veh_final.csv')

#A ce stade Vehicules et usagers sont mergés dans un df qui contient 273226 lignes

```

```

#####
##### Travail table lieux #####
import pandas as pd
df_lieux=pd.read_csv("/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/all_data_lieux_2019.csv")

# On constate dans un premier temps que le df_usag8 précédent, qui contient désormais 1 ligne par véhicule, compte
print('nombre de valeurs id_vehicules uniques pour df_usag8',len(df_usag8["id_vehicule"].unique()))
print('nombre de valeurs id_vehicules uniques pour df_veh',len(df_veh["id_vehicule"].unique()))

# On crée 2 df pour me permettre de comparer et identifier les véhicules qui n'apparaissent pas dans la table usagers
id_veh=df_veh[["Num_Acc","id_vehicule"]]
id_veh
id_veh_usagers=df_usag8[["Num_Acc","id_vehicule"]]
```

```

id_veh_usagers["compare"]=1
id_veh_usagers
df_compare=id_veh.merge(id_veh_usagers,on=["Num_Acc","id_vehicule"],how="left")
df_compare.isna().sum()
id_diff=df_compare[df_compare["compare"].isna()]
print(id_diff) # ce sont tous les numéros d'accidents pour lesquels ce numéro de véhicule n'est pas dans ma table us

df_lieux_filtre=df_lieux[["Num_Acc","catr","circ","nbv","vma","surf"]]

df_lieux_filtre

df_lieux_filtre["catr"].value_counts() # 4 représente route communale, 3 route départementale , 1 autoroute et 2 nationales

#####
##### NEW CATR : on choisit de regrouper les nationales/dep/communales d'un côté, les autoroute de l'autre
# #1 Autoroute
# #2 nationale, départementale et communale 2,3,4
# #3 Autre 5,6,7,9

df_lieux_filtre["autoroute"]=0
df_lieux_filtre["nationale_departementale_communale"]=0
df_lieux_filtre["autre_route"]=0
def replace_catr(row):
    if row["catr"] in [1]:
        row["autoroute"] +=1
    if row["catr"] in [2,3,4]:
        row["nationale_departementale_communale"] +=1
    if row["catr"] in [5,6,7,9]:
        row["autre_route"] +=1
    return row

df_lieux_filtre=df_lieux_filtre.apply(replace_catr,axis=1)

#####
## Sens de circulation " CIRC" qu'on regroupe en 2 modalités #####
df_lieux_filtre["circ"].value_counts()

# #1 – A sens unique (initialement modalité 1)
# #2 – Bidirectionnelle (2 et autres)

df_lieux_filtre["sens_unique"]=0
df_lieux_filtre["bidirectionnel"]=0

def replace_circ(row):
    if row["circ"] in [1]:
        row["sens_unique"] +=1
    else:
        row["bidirectionnel"] +=1
    return row

df_lieux_filtre=df_lieux_filtre.apply(replace_circ,axis=1)

#####
## SURFACES #####
df_lieux["surf"].value_counts() # 1 route normale, 2 route mouillée

# au vu de la distribution, on regroupe comme suit :

```

```

#1 normale
#2 mouillée/enneigée
#3 autre

# lorsqu'il y a 2 types de surfaces renseignés ( ce qui arrive notamment lorsque l'accident est survenu au niveau d'un
# 272 valeurs non renseignées, on les gardes ainsi en les incluant dans "autres"

df_lieux_filtre["route_seche"]=0
df_lieux_filtre["route_mouillee_enneigee"]=0
df_lieux_filtre["etat_route_autre"]=0
def replace_surf(row):
    if row["surf"] in [1]:
        row["route_seche"] +=1
    if row["surf"] in [2,3,4,5]:
        row["route_mouillee_enneigee"] +=1
    if row["surf"] in [6,7,8,9]:
        row["etat_route_autre"]+=1
    return row

df_lieux_filtre=df_lieux_filtre.apply(replace_surf, axis=1)
df_lieux_filtre

## on fait ensuite un groupby par numéro d'accident , pour le nombre de voies(nbv), lorsqu'il y a 2 valeurs différentes

df_lieux_final=df_lieux_filtre.groupby("Num_Acc").agg({"nbv":"max","vma":"max","nationale_departementale_commun": "max"})

# illustration des accidents où le Nombre de voies (nbv) est différents

grouped_df_nbv = df_lieux.groupby('Num_Acc')['nbv'].nunique().reset_index() # 8570 accidents concernés
diff_nbv_groups = grouped_df_nbv[grouped_df_nbv['nbv'] > 1]['Num_Acc']

print(diff_nbv_groups)

# par ailleurs, on constate qu'il y a dans la table lieux initiale, 54 lignes pour lesquelles la colonne NBV est #valeurmultipliée
print(len(df_lieux.loc[df_lieux["nbv"]=="#VALEURMULTI"]))
print(len(df_lieux.loc[df_lieux["nbv"]=="#ERREUR"]))
print(df_lieux.value_counts("nbv"))

# Il y a également 4404 valeurs non renseignées, et des valeurs aberrantes (9, 10, 11, 12 voies, peu probable sur les routes)

##### illustration des accidents où la vitesse max (vma) est différente #####
grouped_df_vma = df_lieux.groupby('Num_Acc')['vma'].nunique().reset_index() # 5169 accidents concernés
diff_vma_groups = grouped_df_vma[grouped_df_vma['vma'] > 1]['Num_Acc']

print(diff_vma_groups)
print(df_lieux_final["vma"].unique()) # il y a quelques valeurs aberrantes ( ex. 300 900 700 800)
print(df_lieux_final.loc[df_lieux_final["vma"]>130]) # ces valeurs aberrantes ( supérieures à 130 qui est la vitesse max
df_lieux_final.info()

##### Sauvegarde le df_lieux_final dans le drive partagé , il contient désormais 273226 lignes , 1 ligne par accident

```

```
df_lieux_final.to_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_lieux_final', index=False)
```

```
#####
# Travail table caractéristiques #####
#####

# importation du df_caractéristiques
df_carac=pd.read_csv("/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/all_data_carac_2019_2023.csv")
df_carac

# Merge de la table lieux précédente avec la table caractéristiques
df_carac_lieux=df_carac.merge(df_lieux_final,on="Num_Acc",how="left")

# Importation de la table précédemment mergée entre usagers et véhicules, puis merge des 2 tables afin d'obtenir le df_total_final
df_merged_usag_veh_final=pd.read_csv("/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/all_data_usag_veh_2019_2023.csv")
df_total_final=df_carac_lieux.merge(df_merged_usag_veh_final,on="Num_Acc",how="left")

# On renomme quelques colonnes pour une meilleure compréhension
df_total_final=df_total_final.rename({"new_catv_1":"VL_VU","new_catv_2":"2roues_3roues","new_catv_3":"PL","new_catv_4":"IMM","new_catv_5":"VH_VU"},axis=1)

# Ajout d'une colonne qui détermine la gravité de l'accident, nous décidons de conserver la gravité la + importante lors d'un accident
df_total_final["gravité_accident"]=0

#4 accident mortel
#3 accident avec blessés hospitalisés
#2 accident avec blessés légers
#1 accident avec des indemnes

def determine_gravite(row):
    if row["tué"]>0:
        return 4
    if row["blessé_hospitalisé"]>0:
        return 3
    if row["blessé_léger"]>0:
        return 2
    if row["indemne"]>0:
        return 1

df_total_final['gravité_accident'] = df_total_final.apply(determine_gravite, axis=1)

# On supprime les colonnes inutiles pour notre étude (adr, com id_vehicule, lum)
# On supprime les colonnes redondantes (atu_conducteur, passager et catu_pieton sont redondantes avec la place de l'accident)
# On supprime les colonnes non renseignées, qui n'apportent aucune information (sexé non renseigné, total_secu_non_renseigné)
# ON exclut les outliers de la variable age (>96)
df_total_final=df_total_final.drop(["adr","com","id_vehicule","lum", "catu_pieton", "catu_conducteur", "passager", "sexé", "total_secu_non_renseigné"],axis=1)

# On enregistre le df_total_final dans le drive partagé
df_total_final.to_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_total_final.csv', index=False)
```

```

#####
##### Usagers #####
#### Variable sexe ####

sex = df.groupby('gravité_accident').agg({'homme':'sum', 'femme': 'sum'}).reset_index()
display(sex)
fig = px.bar(sex, x='gravité_accident', y=['homme','femme'], title="Gravité selon le sexe", barmode='group')
fig.update_xaxes(tickmode='array', tickvals=[2,3,4], ticktext=['blessé_léger', 'blessé_grave', 'tué'])
fig.show();

grav_homme = sex[['gravité_accident', 'homme']]
grav_femme = sex[['gravité_accident','femme']]

labels = ['Blessés légers', 'Blessés hospitalisés', 'Tués']
fig,axs=plt.subplots(1,2,figsize=(20,8))

axs[0].pie(grav_homme['homme'], labels = grav_homme['gravité_accident'], colors=["green","orange","red"], autopct='%1.1f%%')
axs[0].set_title("Hommes")

axs[1].pie(grav_femme['femme'], labels = grav_femme['gravité_accident'], colors=["green","orange","red"], autopct='%1.1f%%')
axs[1].set_title("Femmes")

fig.suptitle("Répartition des accidents par gravité chez les hommes et chez les femmes")
fig.legend(labels, loc='upper left')

#####
##### Age #####
####

age = df.groupby('gravité_accident').agg({'0-17': 'sum', '18-60':'sum', '61-95':'sum'}).reset_index()
display(age)
fig = px.bar(age, x='gravité_accident', y=['0-17','18-60','61-95'], title="Répartition du nombre d'usagers par classe d'âge")
fig.update_xaxes(tickmode='array', tickvals=[2,3,4], ticktext=['blessé_léger', 'blessé_grave', 'tué'])
fig.show();

grav_0_17 = age[['gravité_accident','0-17']]
grav_18_60 = age[['gravité_accident','18-60']]
grav_61_95 = age[['gravité_accident','61-95']]

labels = ['Blessés légers', 'Blessés hospitalisés', 'Tués']
fig,axs=plt.subplots(1,3,figsize=(20,8))

axs[0].pie(grav_0_17['0-17'], labels = grav_0_17['gravité_accident'], colors=["green","orange","red"], autopct='%1.1f%%')
axs[0].set_title("0-17 ans")

axs[1].pie(grav_18_60['18-60'], labels = grav_18_60['gravité_accident'], colors=["green","orange","red"], autopct='%1.1f%%')
axs[1].set_title("18-60 ans")

axs[2].pie(grav_61_95['61-95'], labels = grav_61_95['gravité_accident'], colors=["green","orange","red"], autopct='%1.1f%%')
axs[2].set_title("61-95 ans")

fig.suptitle("Répartition des accidents par gravité pour les différentes tranches d'âge")
fig.legend(labels, loc='upper left')

```

```
#####
##### Système de sécurité #####
grav = df.groupby('gravité_accident').agg({'total_sans_secu':'sum', 'total_ceinture': 'sum', 'total_casque': 'sum', 'total_échec': 'sum'})
display(grav)
fig = px.bar(grav, x='gravité_accident', y=[ 'total_sans_secu', 'total_ceinture', 'total_casque', 'total_secu_enfant', 'total_échec'])
fig.update_xaxes(tickmode='array', tickvals=[2,3,4], ticktext=['blessé_léger', 'blessé_grave', 'tué'])
fig.show();
```

```
#####
##### Localisation de l'accident #####
# Carte géographique de la répartition des accidents par gravité
```

```
import geopandas as gpd
from shapely.geometry import Point

# Affichage des coordonnées de chaque accident, et transformation de 'lat', 'long' et 'dep' en type int.
df['lat']= df['lat'].astype('str').apply(lambda x: x.replace('\t', '').replace(';', ''))
df['lat'] = df['lat'].astype('float').round(5)
df['long']= df['long'].astype('str').apply(lambda x: x.replace('\t', '').replace(';', ''))
df['long'] = df['long'].astype('float').round(5) #avec 5 décimales, précision de 1m

# Lignes dont les départements sont numériques
df_dep_num= df.loc[df['dep'].astype('str').apply(lambda x: x.isnumeric())].astype('str')
```

```
# Lignes pour lesquelles le dep est la corse
df_dep_corse = df.loc[df['dep'].astype('str').apply(lambda x: x.isnumeric() == False)].astype('str')
```

```
# Sélectionner les accidents dans les départements dont le code est inférieur à 900, c'est à dire ceux en métropole
df_m = df[(df['dep'].isin(df_dep_num.dep.unique()) | (df['dep'].isin(df_dep_corse.dep.unique())))]
```

```
print(df_m['dep'].unique())
# Supprimer les accidents avec des coordonnées de latitude et de longitude nulles ou manquantes
df_m = df_m[(df_m['lat'] != 0.0) & (df_m['long'] != 0.0)].dropna(subset = ['lat', 'long'], axis = 0)
```

```
# Créer une géométrie Point pour chaque paire de coordonnées (longitude, latitude)
geometry = [Point(xy) for xy in zip(df_m['long'], df_m['lat'])]
```

```
# Créer un GeoDataFrame à partir des données d'accidents et de la géométrie Point
geo_df = gpd.GeoDataFrame(df_m, geometry=geometry)
```

```
# Créer une figure et des axes pour la carte
figure, ax = plt.subplots(figsize = (13, 12))
```

```
# Masquer les axes pour une apparence plus propre
plt.axis('off')
```

```
# Définir les limites des axes pour inclure toute la France
ax.set_xlim([-5.5, 9.5]) # Longitude (Ouest-Est)
ax.set_ylim([41.0, 51.5]) # Latitude (Sud-Nord)
```

```

# Tracer les accidents sur la carte en utilisant différentes couleurs pour chaque gravité
geo_df[geo_df['gravité_accident'] == 2].plot(ax = ax, markersize = 5, color = 'green', label = 'blessés légers')
geo_df[geo_df['gravité_accident'] == 3].plot(ax = ax, markersize = 5, color = 'orange', label = 'blessés graves')
geo_df[geo_df['gravité_accident'] == 4].plot(ax = ax, markersize = 5, color = 'red', label = 'tués')
plt.title('Représentation de la gravité des accidents en France entre 2019 et 2023')
plt.legend();

#####
##### Agglo / Hors agglo #####
#####

# Répartition du nombre d'accidents selon la localisation agglo/ non agglo
sns.countplot(x = 'agg', hue = 'gravité_accident', data = df_filtered)
plt.title("Répartition du nombre d'accidents selon la localisation agglo/hors agglo")
plt.xticks(rotation = 90)
plt.xlabel('localisation')
plt.show();

#####
##### Test du Chi2 #####
#####

contingence_agg = pd.crosstab(df['gravité_accident'], df['agg'], normalize = True)
contingence_agg = pd.DataFrame(contingence_agg).apply(lambda x: round(x, 4)*100, axis = 1)
print(contingence_agg)

# Créer la heatmap avec Plotly Express
sns.heatmap(contingence_agg, annot=True, cmap='Blues')

results = chi2_contingency(contingence_agg)
print ('la statistique du test est', results[0])
print ('la-pvalue du test est', results[1])
# Appliquer le test du Chi-carré

print(p_value)
# Interprétation:

if p_value < 0.05:
    print("La localisation (agglo/hors agglo) influence significativement la gravité de l'accident (p < 0.05).")
else:
    print ("La localisation (agglo/hors agglo) n'influence pas la gravité de l'accident (p >= 0.05).")
# Le type de choc "tonneaux" impacte la gravité significativement ( blessés hospitalisés et tués )

```

```

#####
##### Véhicules #####
#####

#####
##### Catégorie véhicule #####
#####

# extraire tous df accidents où un PL est impliqué 9423 accidents
acc_avec_pl=df.loc[df["PL"]>=1]
acc_avec_pl=acc_avec_pl.groupby("gravité_accident").size()
print("proportion des gravités d'accidents impliquant au moins 1 PL",acc_avec_pl.div(acc_avec_pl.sum()))
# 13% des accidents impliquant au moins 1 PL sont mortels ; 30% avec blessés hospi

# extraire tous les accidents où un 2 ou 3 roues ou quad est impliqué 92608 accidents
acc_avec_2_3roues=df.loc[df["2roues_3roues_quad"]>=1]
acc_avec_2_3roues=acc_avec_2_3roues.groupby("gravité_accident").size()
print("proportion des gravités d'accidents impliquant au moins 1 2roues ou 3 roues ou quad ",acc_avec_2_3roues.div(

```

```

# idem pour Bus_Car
acc_avec_bus_car=df.loc[df["bus_car"]>=1]
acc_avec_bus_car=acc_avec_bus_car.groupby("gravité_accident").size()
print("proportion des gravités d'accidents impliquant au moins 1 bus ou car ", acc_avec_bus_car.div(acc_avec_bus_ca

# idem pour VL VU  222890 accidents

acc_av_vl_vu=df.loc[df["VL_VU"]>=1]
acc_av_vl_vu=acc_av_vl_vu.groupby("gravité_accident").size()
print("proportion des gravités d'accidents impliquant au moins 1 VL VU",acc_av_vl_vu.div(acc_av_vl_vu.sum()))

# idem pour velo trott et edp  45918 accidents
acc_av_velo_trott_edp=df.loc[df["velo_trott_edp"]>=1]
acc_av_velo_trott_edp=acc_av_velo_trott_edp.groupby("gravité_accident").size()
print("proportion des gravités d'accidents impliquant au moins 1 engin de déplacement personnel (velo, trott ou edp)",

fig,axs=plt.subplots(1,5,figsize=(15,10))
labels = ['Blessés légers', 'Blessés hospitalisés', 'Tués']

axs[0].pie(acc_avec_pl,colors=["green","orange","red"],autopct="%1.1f%%")
axs[0].set_title("Avec PL")
axs[1].pie(acc_avec_2_3roues,colors=["green","orange","red"],autopct="%1.1f%%")
axs[1].set_title("Avec 2 ou 3 roues ou quad")
axs[2].pie(acc_avec_bus_car,colors=["green","orange","red"],autopct="%1.1f%%")
axs[2].set_title("Avec bus/car")
axs[3].pie(acc_av_vl_vu,colors=["green","orange","red"],autopct="%1.1f%%")
axs[3].set_title("Avec VL VU")
axs[4].pie(acc_av_velo_trott_edp,colors=["green","orange","red"],autopct="%1.1f%%")
axs[4].set_title("Avec velo, trott ou edp")

fig.suptitle("Répartition des accidents par gravité pour différents types de véhicules")
fig.legend(labels, loc='center')

# idem pour piétons  44692 accidents
acc_av_pietons=df.loc[df["place_pieton"]>=1]
acc_av_pietons=acc_av_pietons.groupby("gravité_accident").size()
print("proportion des gravités d'accidents impliquant au moins 1 piéton",acc_av_pietons.div(acc_av_pietons.sum()))

```

```

#####
##### Matrices de corrélation #####
#####

# Pour plus de lisibilité, on fait plusieurs heatmaps sur des sélections de colonnes

#####
##### Matrice de corrélation usagers #####
#####

selected_columns1 = [ 'usager_count', '0-17' , '18-60', '61-95', 'gravité_accident','homme','femme', 'place_conducteur',
corr_matrix_subset1 = df_filtré[selected_columns1].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix_subset1, annot=True, cmap="coolwarm", linewidths=0.5, fmt=".2f")
plt.show()

#####
##### Matrice de corrélation selon les conditions de la route #####
#####

import numpy as np
df_filtré['nbv'] = df_filtré['nbv'].replace(to_replace = ['#ERREUR', '#VALEURMULTI'], value = np.nan)

```

```

selected_columns3 = ['gravité_accident','nationale_departementale_communale', 'autoroute', 'autre_route', 'sens_unique']
corr_matrix_subset3 = df_filtré[selected_columns3].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix_subset3, annot=True, cmap="coolwarm", linewidths=0.5, fmt=".2f" )
plt.show()

#####
# Matrice de corrélation: obstacles et types de chocs #####
#####

selected_columns2 = ['gravité_accident', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc','choc_AV','choc_AR', 'choc_c']
corr_matrix_subset2 = df_filtré[selected_columns2].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix_subset2, annot=True, cmap="coolwarm", linewidths=0.5, fmt=".2f")
plt.show()

```

## 2. Modélisation

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

- a. Feature engeneering pour l'élaboration d'un dataframe avec les variables sélectionnées , nommé df\_machine learning.
  - b. Travail sur les séries temporelles
  - c. Modèle 3 classes Random Forest
  - d. Modèle 3 classes Régression Logistique
  - e. Modèle 3 classes KNN
  - f. Modèle 3 classes XGBoost
  - g. Modèle 2 classes Random Forest
  - h. Modèle 2 classes Régression Logistique
  - i. Modèle 2 classes KNN
  - j. Modèle 2 classes XGBoost

a. Feature engineering pour l'élaboration 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 permet d'avoir un fichier mis à jour si modifié

file_path = '/content/drive/My Drive/Datascientest/Projet_accidents/Dataset/2019_2023/df_total_final.csv'
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

```



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

```
df['date']= pd.to_datetime(df['jour'].astype('str')+'/' +df['mois'].astype('str')+'/' +df['an'].astype('str')+'/' +df['hrmn'].ast
df['heure'] = df['date'].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, '10': 10, '9': 9})
df.nbv.value_counts()
```

```

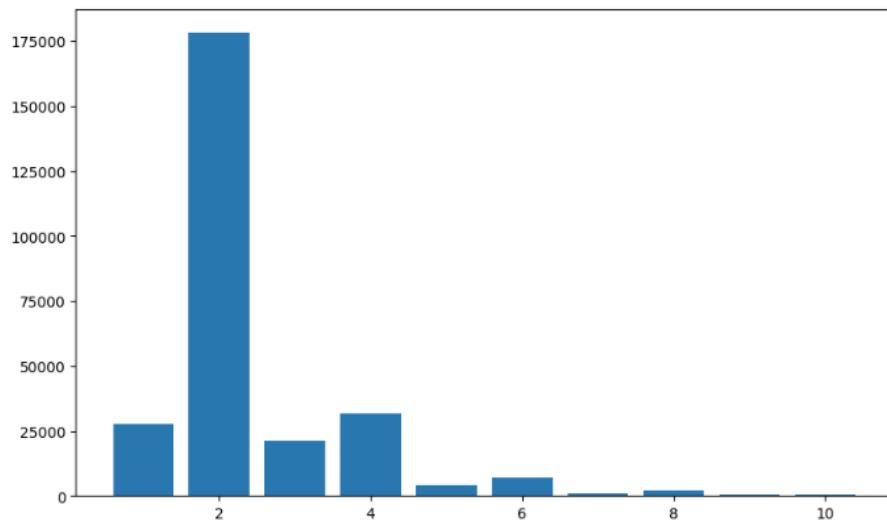
count
nbv
2    178149
4    31627
1    27445
3    21215
6    6864
5    4219
8    2125
7    755
10   510
9    317
dtype: int64

```

```

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();

```

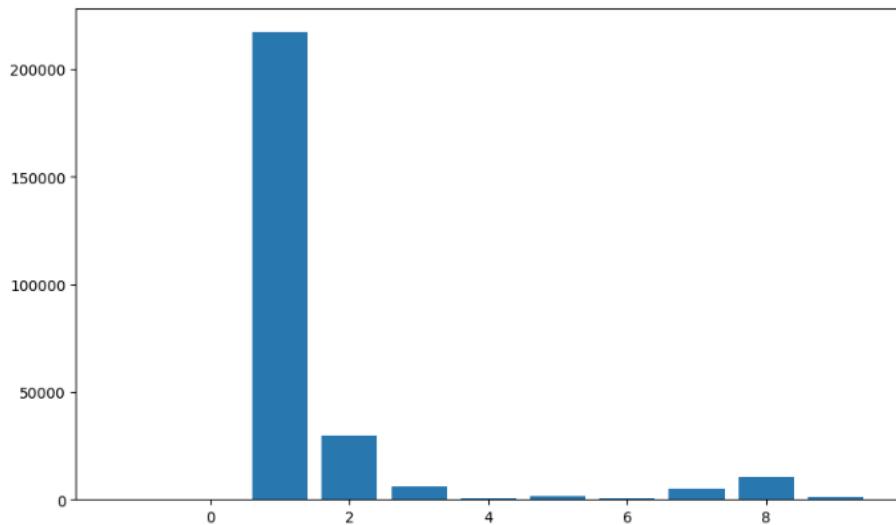


Nettoyage colonne atmosphère (atm): on les regroupe en 4 catégories: 'Temps normal', 'temps couvert', 'temps pluvieux', 'autre'

```

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()
```

atm	count
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/2019_2023/df_machine_learning.csv', index=False)
```

## b. 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 permet d'avoir un fichier mis à jour si modifié

file_path = '/content/drive/My Drive/Datascientest/Projet_accidents/Dataset/2019_2023/df_total_final.csv'
df_total_final=pd.read_csv("/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_total_final.csv")
df_temp = df_total_final.copy()
df_temp['date']= pd.to_datetime(df_temp['jour'].astype('str')+'/'+'mois'].astype('str')+'/'+'an'].astype('str')
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','dep']]

print(df_temp.head())
# On enregistre le df_total_final dans le drive partagé
df_temp.to_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_temp.csv', index=False)

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 × 1 columns
dtype: int64

```

```

#Il y a 25239 accidents dans le département 75, 13955 dans le 93, 12353 dans le 13, données suffisantes pour effectuer une analyse temporelle
#Nous allons tester des prédictions par analyse temporelle, sur le département 75.
#Cette analyse peut s'appliquer à tous les départements (s'ils ont suffisemment d'accidents recensés pour effectuer une analyse temporelle)

```

# Filtrage pour le département 75

```

df_temp = pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_temp.csv', index=False)
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

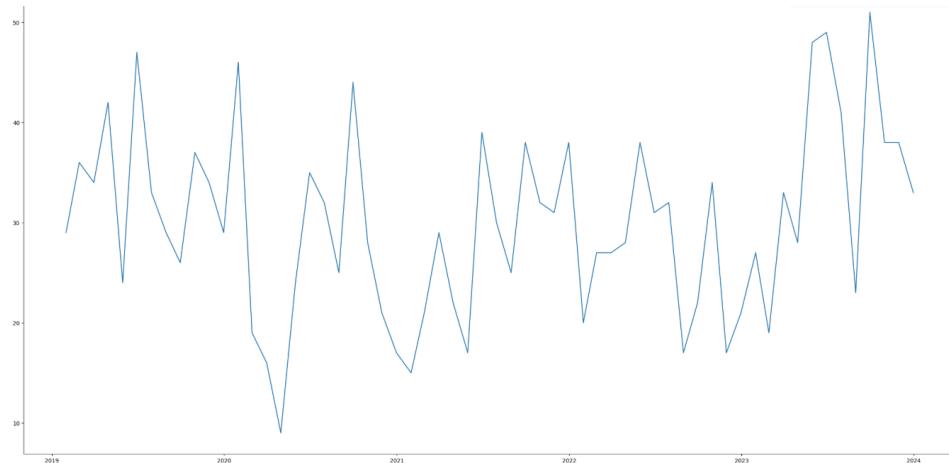
```

```
# 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()

```

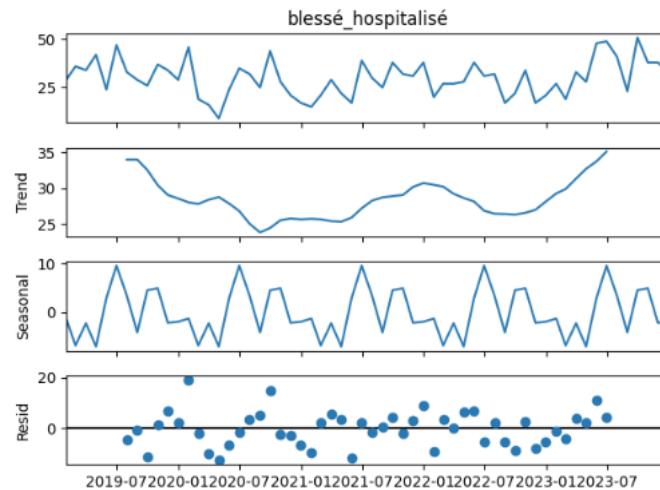


```
from statsmodels.tsa.seasonal import seasonal_decompose
```

```

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. Testons un modèle de régression linéaire.
```

```
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 sur les 3 derniers mois)
```

```
#Hospitalisé
df_temp_75_resampled['blessé_hospitalisé_MA_3months']= df_temp_75_resampled['blessé_hospitalisé'].rolling(window=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 sur les 3 derniers mois)
```

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

```
# Suppression des Nans
```

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

an_mois_jour	blessé_léger	blessé_hospitalisé	tué	blessé_hospitalisé_lag1	blessé_hospitalisé_MA_3months
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:]

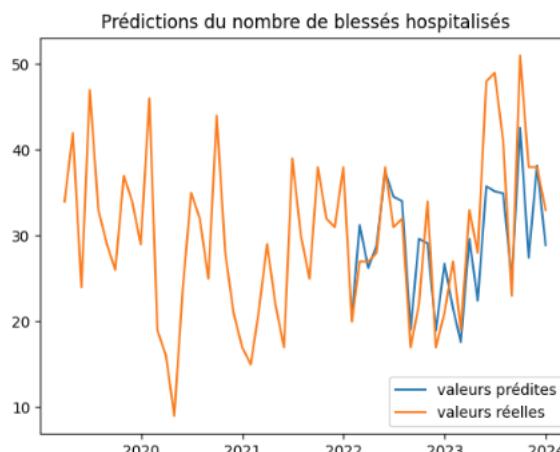
y_hospitalisé_train = y_hospitalisé[:-24]
y_hospitalisé_test = y_hospitalisé.iloc[-24:]

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_test))

plt.plot(y_hospitalisé_test.index, pred_test_hospitalisé, label = 'valeurs prédites')
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.

### c. 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 permet d'avoir un fichier mis à jour si modifié
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_machine_learning.csv')
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_matrix, classification_report, roc_auc_score
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', 'col', 'tué', 'blessé_léger', 'indemne', 'blé'])
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_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 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),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))])

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

# Prédictions
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=['léger', 'hospitalisé', 'tué'], yticklabels=['léger', 'hospitalisé', 'tué'])
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['classifier']

# 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 encodées
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_feature_names_out(categorical_features)
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la transformation de l'heure

# 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 : 0.69
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

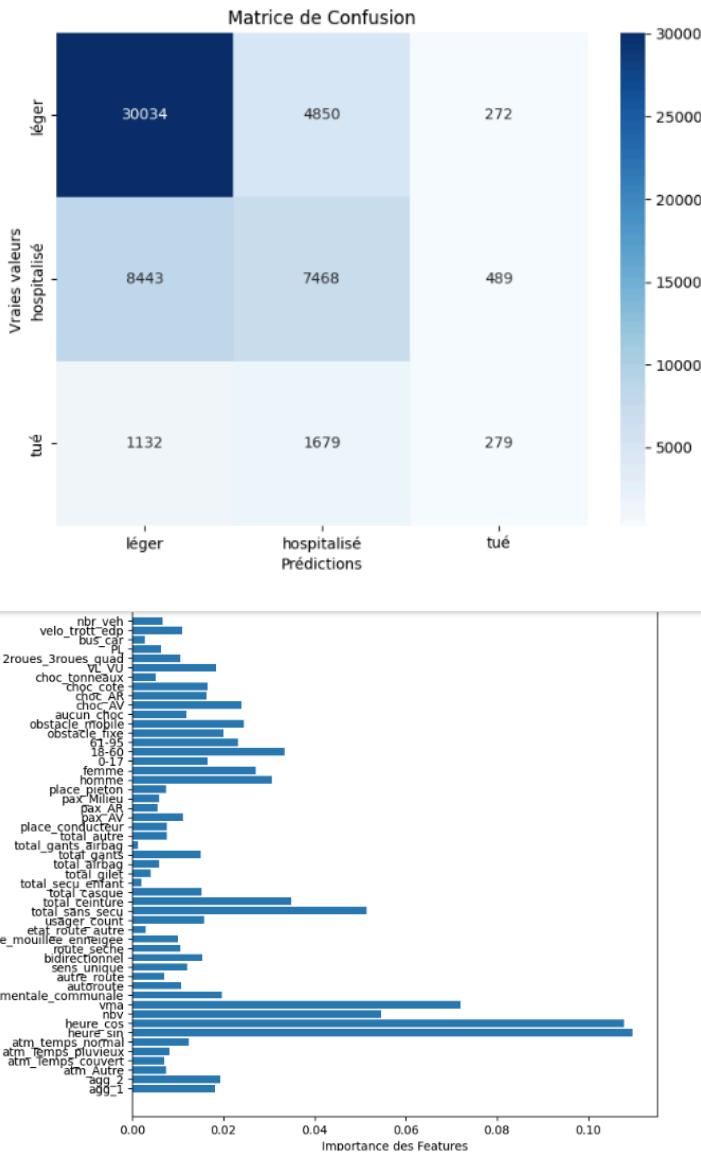
      accuracy                           0.69    54646
      macro avg       0.52     0.47     0.48    54646
  weighted avg       0.66     0.69     0.67    54646

```

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', 'autre_route',
             'sens_unique', 'bidirectionnel', 'route_seche', 'atm',
             'route_mouilee_enneigee', 'etat_route_autre', 'usager_count','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'] # 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),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))])

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

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

```

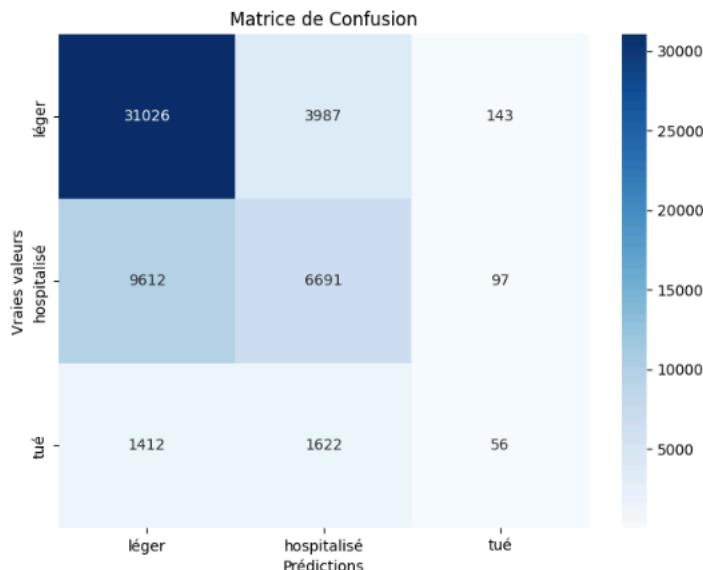
```

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
          macro avg : 0.49
weighted avg : 0.65

```



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', 'col', 'tué', 'blessé_léger','indemne', 'blé', 'victime', 'victime_tuée'], axis=1)
y = dff['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', 'autoroute', 'autre_route',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillée_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 inchangées

# 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', RandomForestClassifier(n_estimators=100, random_state=42)) # Classificateur
])

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

# Prédictions
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=['léger', 'hospitalisé', 'tué'], yticklabels=['léger', 'hospitalisé', 'tué'])
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['classifier']

# 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 encodées
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_feature_names_out(categorical_features)
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la transformation de l'heure

# 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()

```

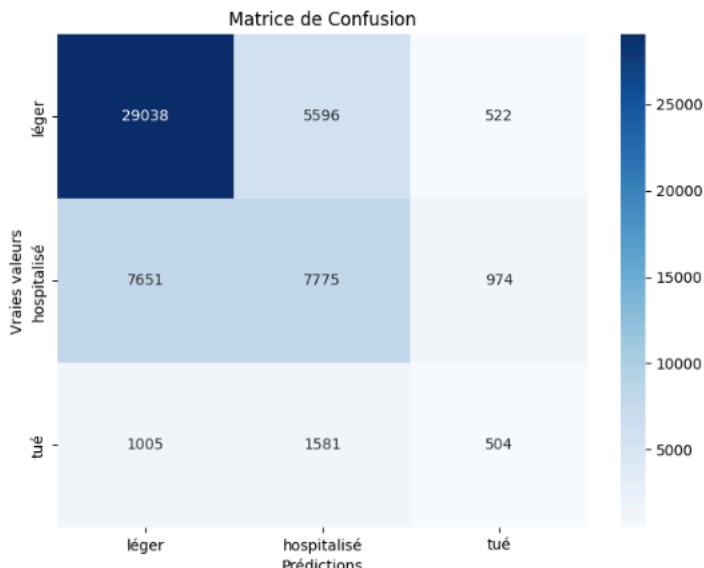
```

Accuracy du modèle 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
          macro avg       0.51     0.49     0.50
  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', 'col', 'tué', 'blessé_léger', 'indemne', 'blé', 'gravité_accident'])
y = dff['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', 'autoroute', 'autre_route',
    '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 inchangées

# 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
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42)) # Classificateur
])

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

# Prédictions
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=['léger', 'hospitalisé', 'tué'], yticklabels=['léger', 'hospitalisé', 'tué'])
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['classifier']

# 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 encodées
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_feature_names_out(categorical_features)
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la transformation de l'heure

# 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 undersampling")
plt.show()

```

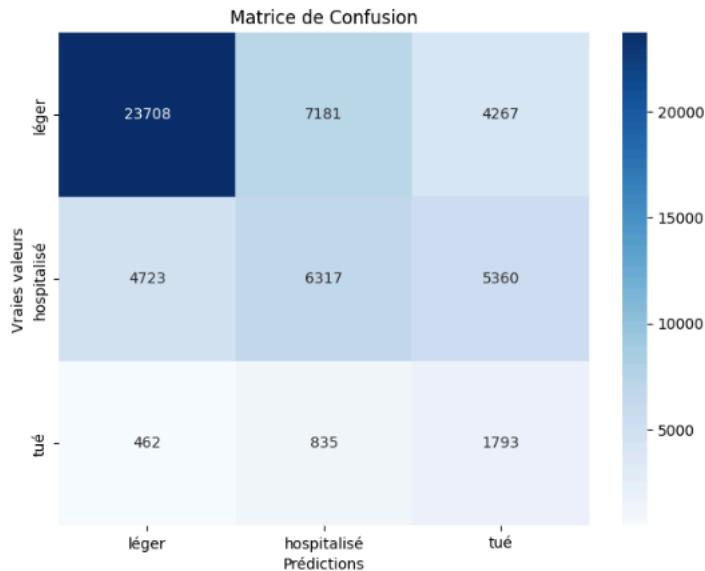
```

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 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', 'blessé_léger', 'indemne', 'blessé_hospitalisé'])
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 inchangées

# ❤️ **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, class_weight='balanced')

# ❤️ **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
    'classifier__class_weight': ['balanced', None] # Tester 'balanced' et None
}

# ❤️ **Lancer la GridSearchCV**
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='f1_weighted', verbose=2, n_jobs=-1)
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))

```

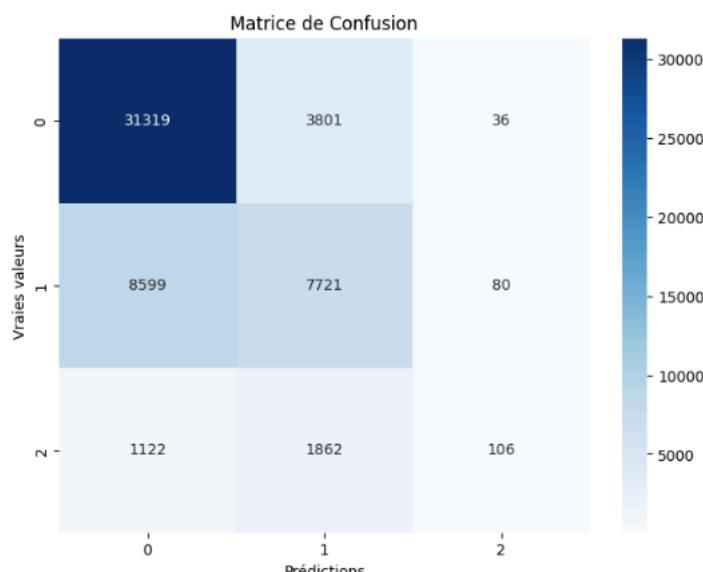
```
# ❤ **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
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      macro avg      weighted avg
0       0.72     0.61        0.69
1       0.47     0.47        0.47
2       0.06     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': 50}

# ON va créer une métrique personnalisée qui donne plus de poids à la prédiction '3': blessé hospitalisé

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', 'col', 'tué', 'blessé_léger', 'indemne', 'bleu'])
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_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 inchangées

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', 'col', 'tué', 'blessé_léger', 'indemne', 'bleu', '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_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', 'autoroute', 'autre_route',
'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 inchangées

# 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)) # Classificateur
])

# 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 classes
    f1_scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring=lambda est, X, y: weighted_f1_score(y, est.predict(X)))
    
    # 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 RandomForest
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
    ))
])

# Entrainer 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.unique(y), yticklabels=np.unique(y))
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")

```

```
plt.title("Matrice de Confusion")
plt.show()
```

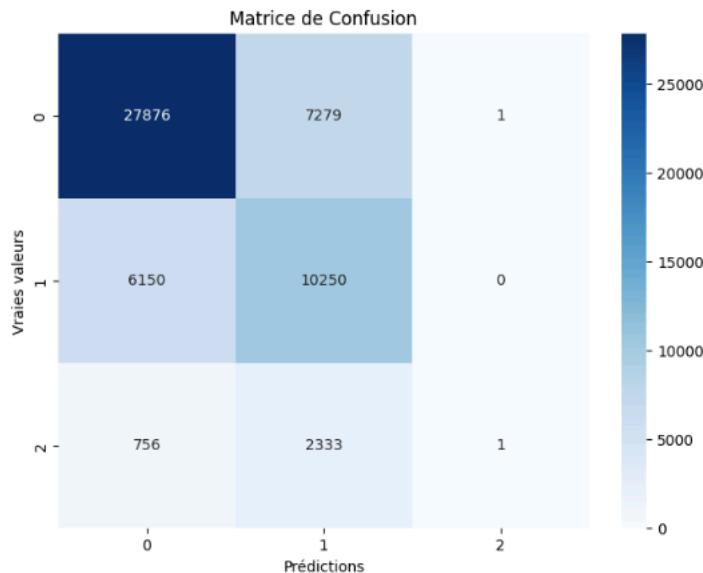
```
Configuration des poids {0: 1, 1: 2, 2: 1} -> Précision pondérée : 0.44670245271593345
Configuration des poids {0: 1, 1: 10, 2: 5} -> Précision pondérée : 0.3539732055397752
Configuration des poids {0: 1, 1: 10, 2: 10} -> Précision pondérée : 0.27334463537080333

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
  macro avg                           0.61
weighted avg                          0.70
```



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_class
    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 précédent)
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 calculer le rappel pondéré
for class_weights in class_weights_list:
    # Utiliser cross_val_score pour évaluer la performance avec les poids de classes
    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_weights_recall}")

# Créer la pipeline en utilisant les poids optimaux dans le classificateur RandomForest
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
    ))
])

# Entrainer 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.unique(y), yticklabels=np.unique(y))
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

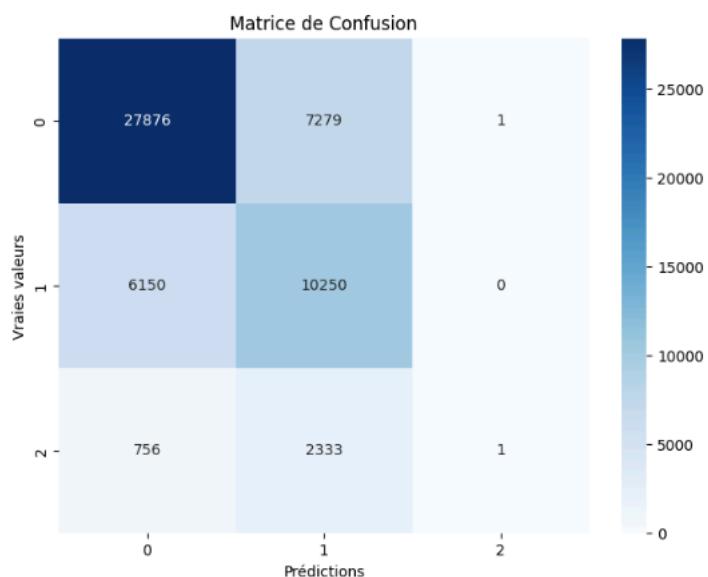
Meilleure configuration de pondération (Rappel) : {0: 1, 1: 2, 2: 1}

📊 **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 grid 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}

#### d. 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 permet d'avoir un fichier mis à jour si modifié
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_machine_learning.csv')
df['gravité_accident'] = df['gravité_accident']-2
df.gravité_accident.value_counts()

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

	count
gravité_accident	
0	175525
1	82229
2	15472

## # 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, classification_report
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', 'col', 'tué', 'blessé_léger', 'indeemne', 'blé'])
```

```

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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 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 avec la régression logistique
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000, random_state=42))]) # Régression Logistique avec itérations supplémentaires

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

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

```

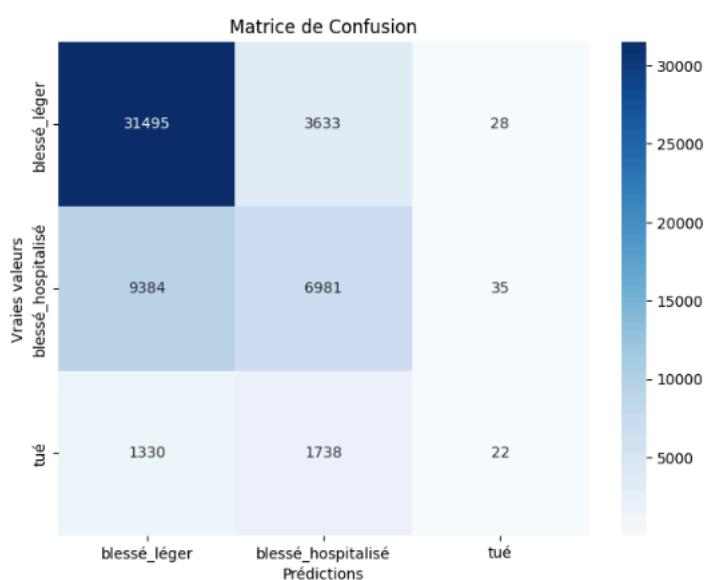
```

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', 'col', 'tué', 'blessé_léger','indemne', 'ble
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 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 avec SMOTE (oversampling) et la régression logistique
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('smote', SMOTE(random_state=42)), # Oversampling avec SMOTE
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression logistique
])

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

```

# Prédictions
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=['blessé_léger','blessé_hospitalisé', 'tué'], y
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

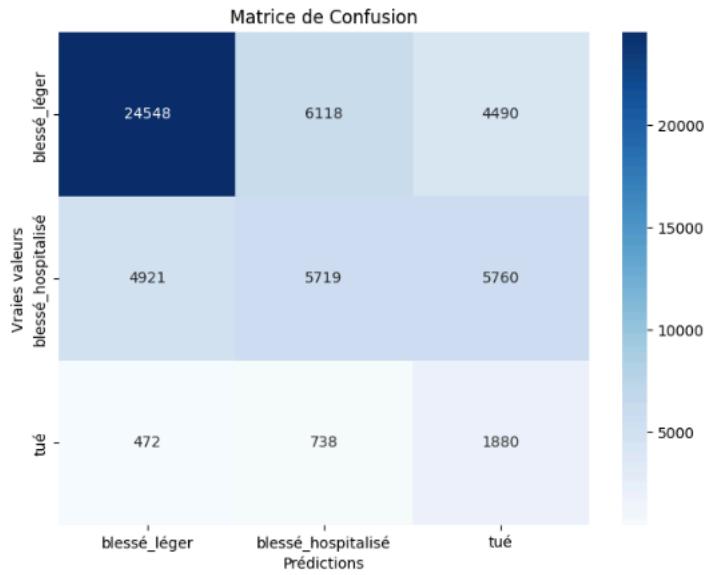
```

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
  macro avg                           0.48
weighted avg                          0.67

```



Recall classe 1: 0.35

F1\_Score classe 1: 0.39

L'oversampling dégrage 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', 'col', 'tué', 'blessé_léger','indemne', 'ble
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 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 avec RandomUnderSampler (undersampling) et la régression logistique
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling avec RandomUnderSampler
    ('classifieur', LogisticRegression(max_iter=1000, random_state=42)) # Régression logistique
])

# 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=['blessé léger', 'blessé hospitalisé', 'tué'], y
plt.xlabel('Prédiction')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

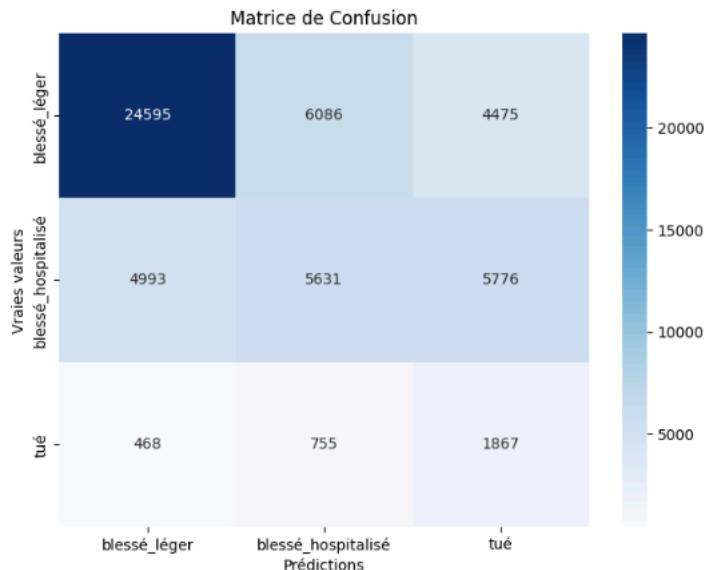
```

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
macro avg                         0.47
weighted avg                      0.67

```



Recall classe 1: 0.34

F1\_Score classe 1: 0.39

De même, l'undersampling dégrade les scores.

Gridsearch pour optimisater 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', 'col', 'tué', 'blessé_léger','indemne', 'ble
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', 'autoroute', 'autre_route',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillée_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 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 avec la régression logistique
pipeline = Pipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('classifier', LogisticRegression(max_iter=1000, random_state=42, class_weight = 'balanced')) # Régression logistique
])

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

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

# Entrainer 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))

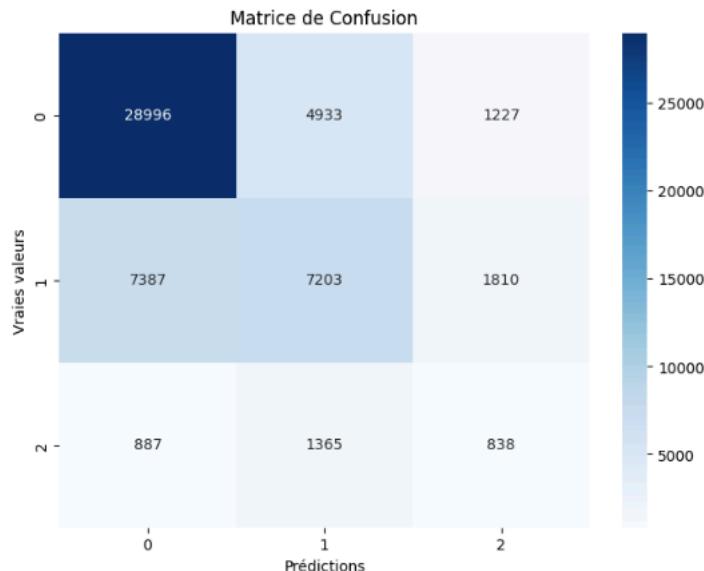
# ❤ **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")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

	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	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**

### e. Modèle 3 classes KNN

Modèle KNN avec Gridserach

```
# 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_score, confusion_matrix, classification_report
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', 'col', 'tué', 'blessé_léger', 'indemne', 'blé', 'y'])
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_state=42)

# Définition des caractéristiques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale', 'autoroute', 'autre_route',
'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', 'fem',
'18-60', '61-95', 'obstacle_fixe', 'obstacle_mobile', 'aucun_choc', 'choc_AV', 'choc_AR', 'choc_cote', 'choc_tonneau',
'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': ['uniform', 'distance']}

# 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', n_jobs=-1)
grid_search.fit(X_train, y_train)

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

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

```

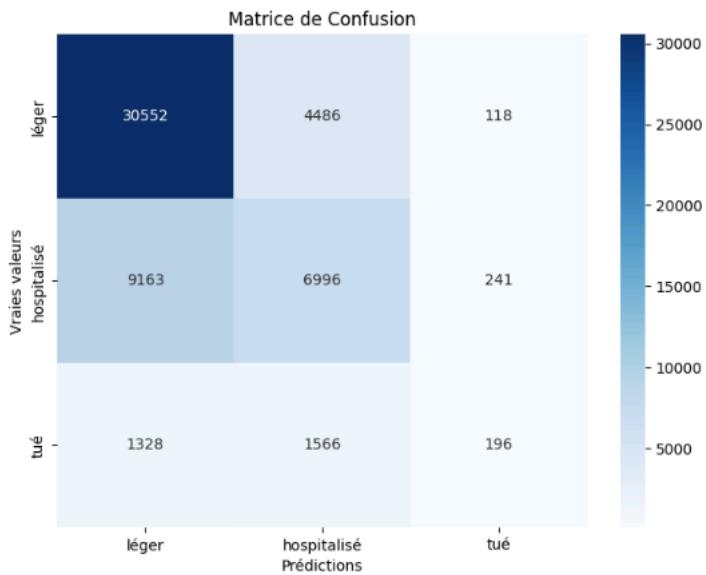
```

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
   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', 'blessé_léger',
            '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, stratify=y)

# Variables catégorielles et cycliques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale', 'autoroute', 'autre_route',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillée_enneigée', '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']

# 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_scoring = make_scoring(f1_class_specific, greater_is_better=True, target_class=1)

# =====
# 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=[0,1,2], yticklabels=[0,1,2])
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", "tué"]))

```

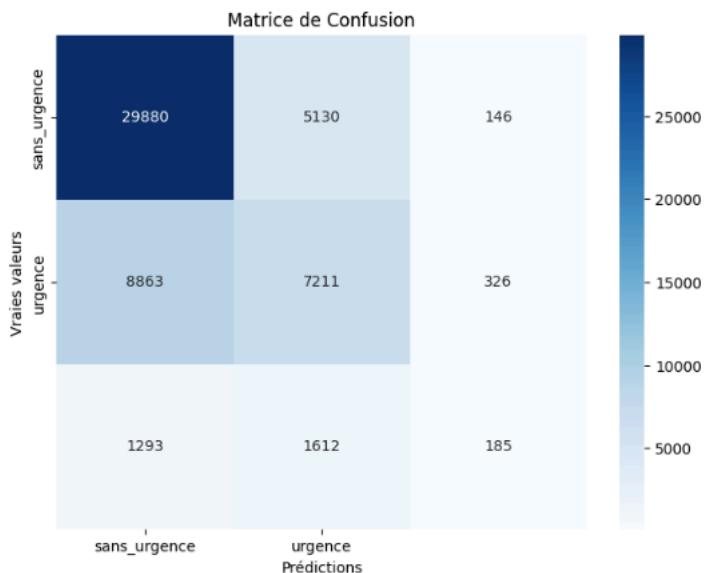
```

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
      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.**

#### **f. 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 permet d'avoir un fichier mis à jour si modifié
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_machine_learning.csv')
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 d'imblearn
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', 'blessé_léger',
             'indemne', 'blessé_hospitalisé', 'gravité_accident'], axis=1)
y = df['gravité_accident']

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

# 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', 'autoroute', 'autre_route',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillée_enneigée', '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 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
```

```

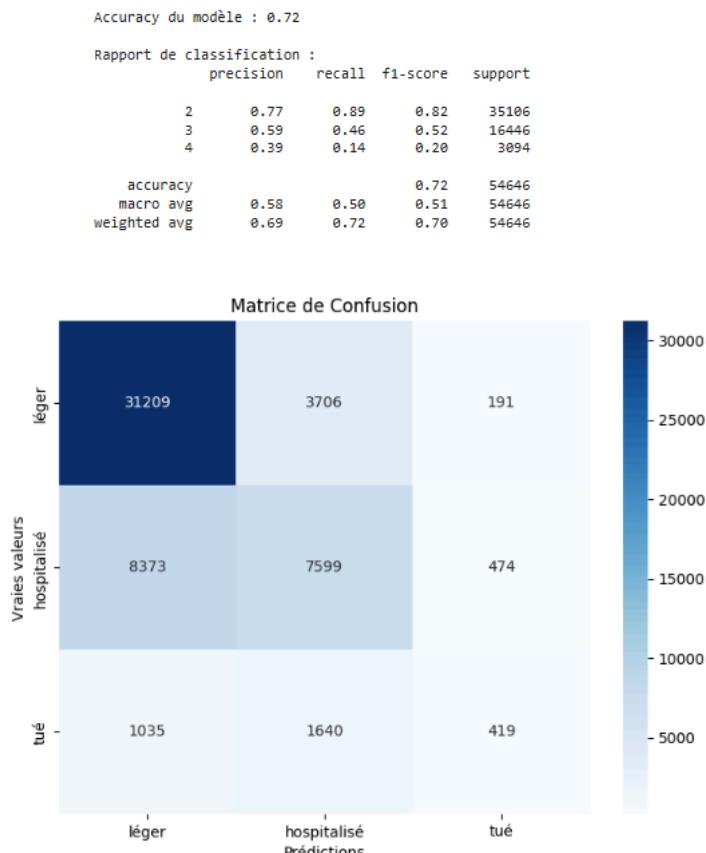
model = XGBClassifier()
objective='multi:softmax', # Classification multiclasses
num_class=3, # Nombre de classes
eval_metric='mlogloss', # Log loss
use_label_encoder=False, # Suppression d'un avertissement lié à XGBoost
random_state=42
)
model.fit(X_train_resampled, y_train_resampled)

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

# Évaluation du modèle
print(f"Accuracy du modèle : {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=['léger', 'hospitalisé', 'tué'],
            yticklabels=['léger', 'hospitalisé', 'tué'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```



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', 'blessé_léger',
             '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] → [0,1,2])
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', 'autoroute', 'autre_route',
                        'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillée_enneigée', '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 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 multiclasses
    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)

# Prédictions (on reconvertit 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 aux vraies classes
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=['léger', 'hospitalisé', 'tué'],
            yticklabels=['léger', 'hospitalisé', 'tué'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

```

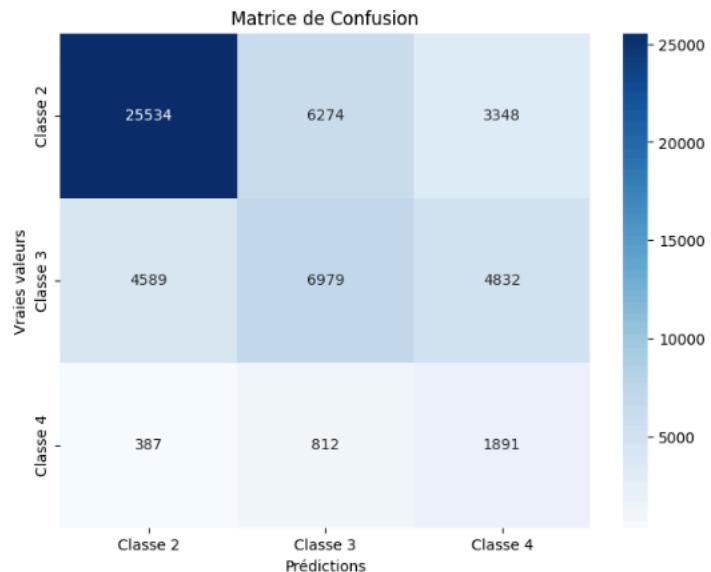
```

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

```

```

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

# Liste des variables catégorielles et numériques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
numerical_features = ['nbv', 'vma', 'nationale_departementale_communale', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche', 'route_mouillée_enneigée', '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']

# 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),
    ('classifier', XGBClassifier(objective='multi:softmax', num_class=3, eval_metric='mlogloss', random_state=42))
])

# 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 classes
}

# Recherche des meilleurs paramètres avec GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='accuracy', n_jobs=-1, verbose=1)

# 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_score_)

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

# Rapport de classification
print("\nRapport de classification :")

```

```

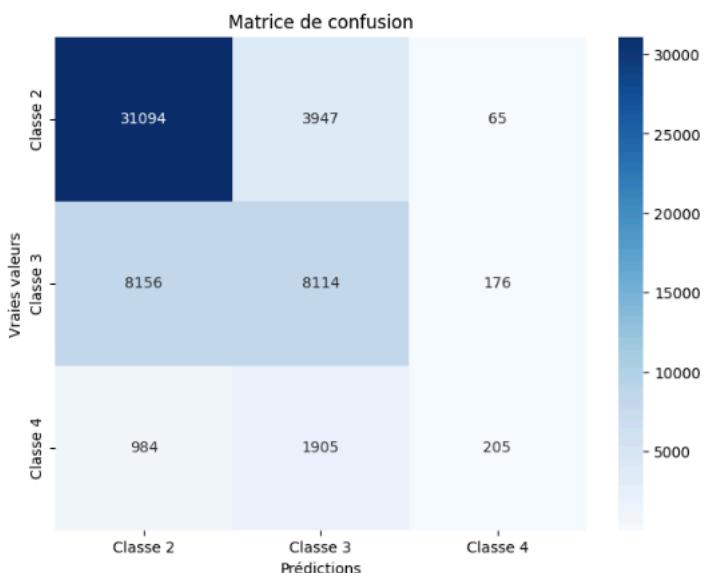
print(classification_report(y_test, y_pred))

# 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=['Classe 2', 'Classe 3', 'Classe 4'], yticklabels=['Vraies valeurs', 'Classe 2', 'Classe 3', 'Classe 4'])
plt.xlabel('Prédictions')
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

	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', 'blessé_léger',
             '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, stratify=y)

# Variables catégorielles et cycliques
categorical_features = ['agg', 'atm']
cyclical_features = ['heure']
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale', 'autoroute', 'autre_route',
                        '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']

# 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_{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_scoring = make_scoring(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_scoring,
    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

```

```

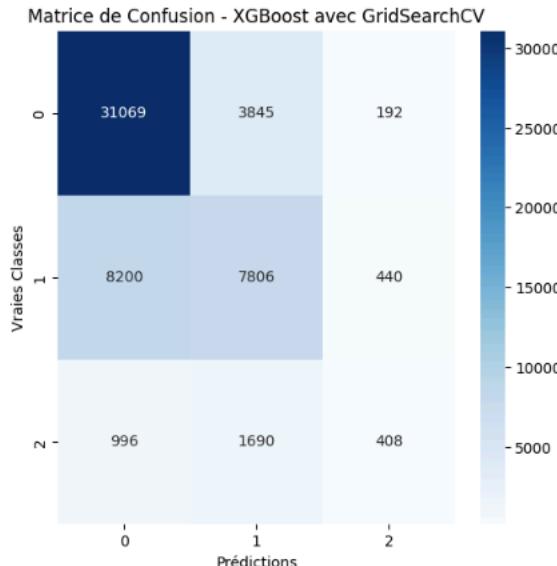
# =====
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=[0,1,2], yticklabels=[0,1,2])
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", "tué"]))

```

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



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)

### g. 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 permet d'avoir un fichier mis à jour si modifié
df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_machine_learning.csv')
df['gravité_accident'] = df['gravité_accident'].replace({2: 'sans_urgence', 3: 'urgence', 4: 'sans_urgence'})
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, classification_report, roc_auc_score
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', 'col', 'tué', 'blessé_léger', 'indemne', 'blé'])
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 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),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))])

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

# Prédictions
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=['Classe 0', 'Classe 1'], yticklabels=['Classe 0', 'Classe 1'])
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['classifier']

# 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 encodées
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_feature_names_out(categorical_features)
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la transformation de l'heure

# 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()

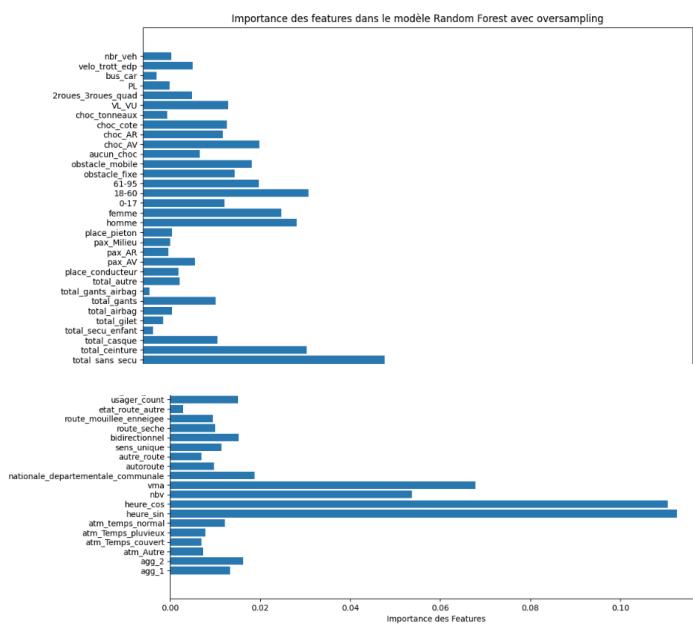
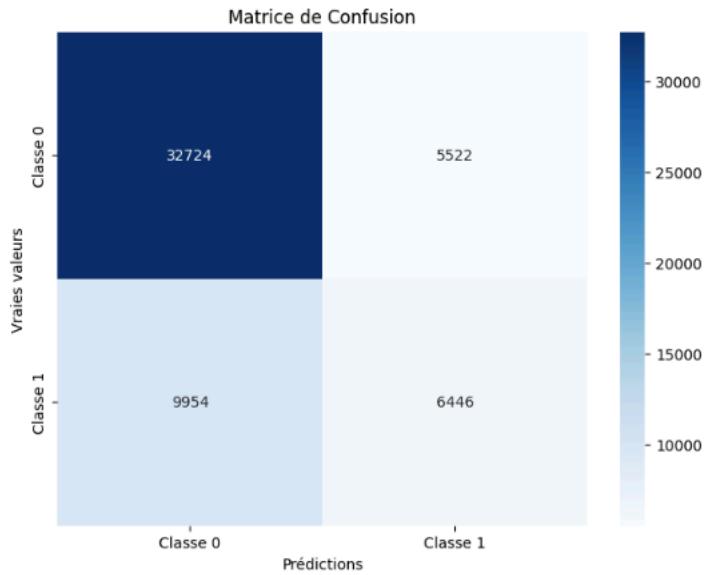
```

```

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          -         -         -      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', 'autre_route',
       'sens_unique', 'bidirectionnel', 'route_seche', 'atm',
       'route_mouillee_enneigee', 'etat_route_autre', 'usager_count','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'] # 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),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))])

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

# Prédictions
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=['Classe 0', 'Classe 1'], yticklabels=['Classe 0', 'Classe 1'])
plt.xlabel('Prédictions')
plt.ylabel('Vraies valeurs')
plt.title('Matrice de Confusion')
plt.show()

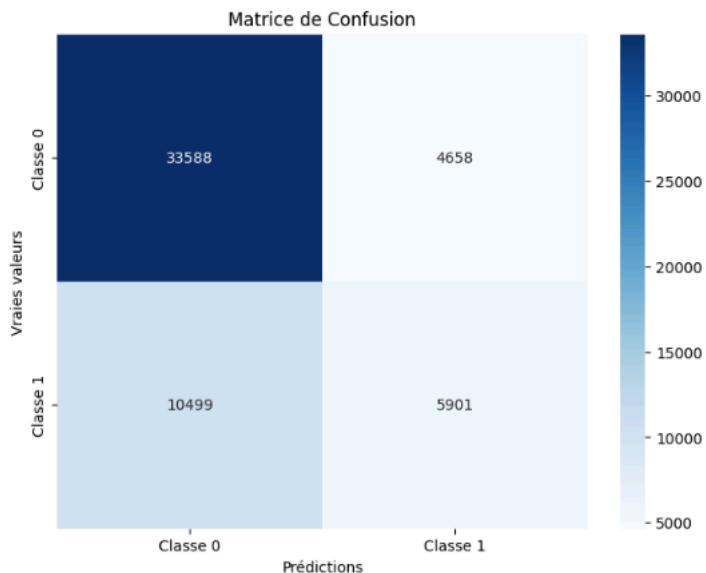
```

```

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          -        -        -    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', 'col', 'tué', 'blessé_léger', 'indemne', 'blk']

```

```

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', 'autoroute', 'autre_route',
'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 inchangées

# 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', RandomForestClassifier(n_estimators=100, random_state=42)) # Classificateur
])

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

# Prédictions
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=['Classe 0', 'Classe 1'], yticklabels=['Classe 0', 'Classe 1'])
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['classifier']

# 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 encodées
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_feature_names_out(categorical_features)
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la transformation de l'heure

# 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()

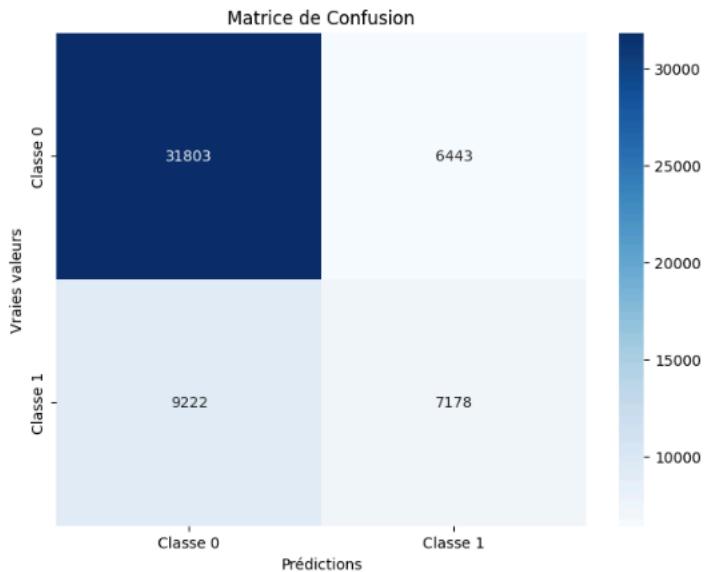
```

```

Accuracy du modèle 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          -        -        -    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', 'col', 'tué', 'blessé_léger', 'indemne', 'bleu'])
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', 'autoroute', 'autre_route',
'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 inchangées

# 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
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42)) # Classificateur
])

# 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"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=['Classe 0', 'Classe 1'], yticklabels=['Classe 0', 'Classe 1'])
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['classifier']

# 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 encodées
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_feature_names_out(categorical_features)
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la transformation de l'heure

# 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 undersampling")

```

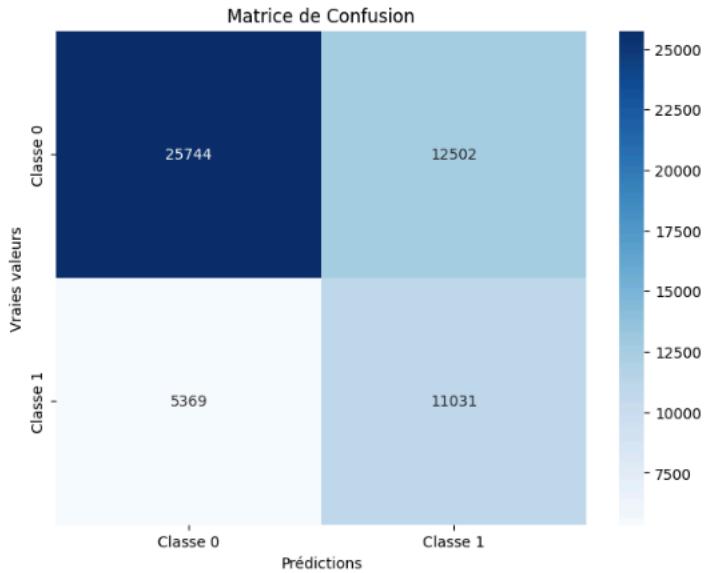
```

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   macro avg   weighted avg
                     0.65     0.67     0.65    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', 'blessé_léger', 'indemne', 'blessé_hospitale'])
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 inchangées

# ❤️ **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, class_weight='balanced')

# ❤️ **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
    'classifier__class_weight': ['balanced', None] # Tester 'balanced' et None
}

# ❤️ **Lancer la GridSearchCV**
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='f1_weighted', verbose=2, n_jobs=-1)
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**

```

```

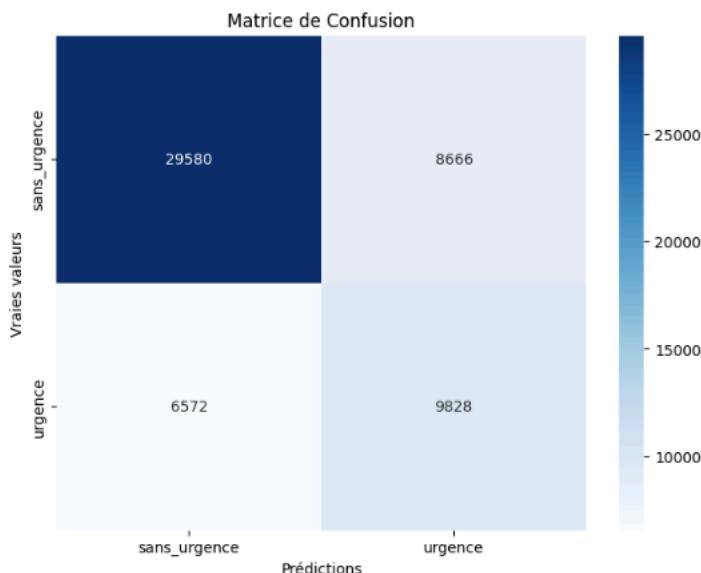
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()

```

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

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

	precision	recall	f1-score	support
sans_urgence	0.82	0.77	0.80	38246
urgence	0.53	0.60	0.56	16480
accuracy			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', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche', 'atm',
'route_mouillée_enneigée', 'etat_route_autre', 'usager_count','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)

# ❤️ **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', verbose=2, n_jobs=-1)
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))

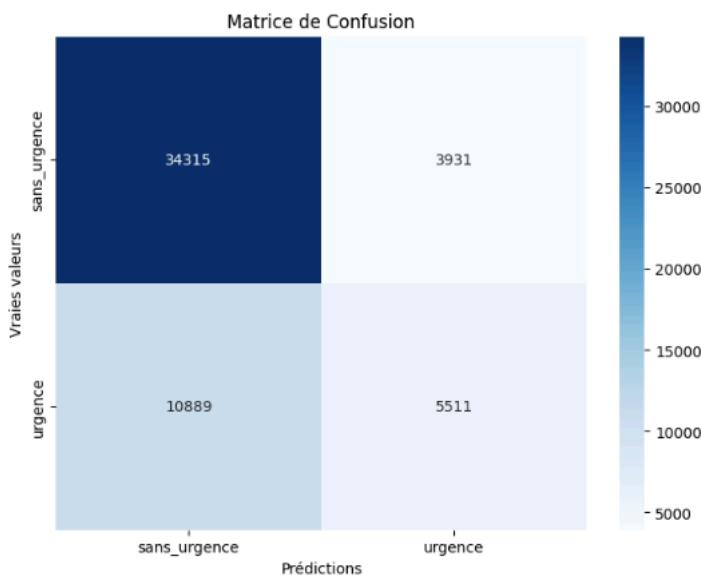
# ❤️ **Matrice de confusion**
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")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

	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

Les scores sont 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', 'col', 'tué', 'blessé_léger', '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_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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 inchangées

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', 'col', 'tué', 'blessé_léger', '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', 'autoroute', 'autre_route',
'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 inchangées

# 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)) # Classificateur
])

# 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 classes
    f1_scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring=lambda est, X, y: weighted_f1_score(y, est.predict(X)))
    
    # 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 RandomForest
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
    ))
])

# Entrainer 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_optimal:.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.unique(y), yticklabels=np.unique(y))
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")

```

```
plt.title("Matrice de Confusion")
plt.show()
```

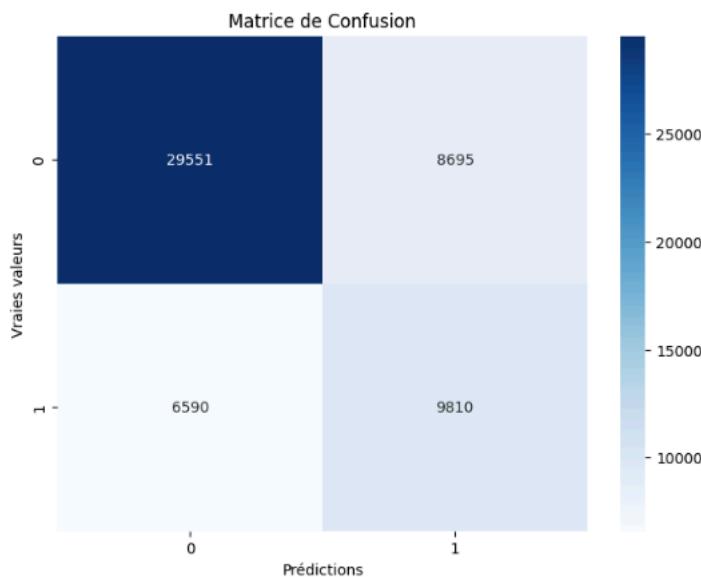
```
Configuration des poids {0: 1, 1: 2} -> Précision pondérée : 0.5588953910759638
Configuration des poids {0: 1, 1: 3} -> Précision pondérée : 0.5254302035131241
Configuration des poids {0: 1, 1: 5} -> Précision pondérée : 0.4919650159502845
Configuration des poids {0: 1, 1: 10} -> Précision pondérée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

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


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



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_score, recall_score, classification_report
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', 'blessé_léger', 'indemne', 'blessé_hospita'])
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', 'autoroute', 'autre_route',
    'sens_unique', 'bidirectionnel', 'route_seche',
    'route_mouillée_enneigée', '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 inchangées

# ❤️ **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 (hospitalisé)**
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_scoring = make_scoring(class_specific_f1, class_index=1)

# ❤️ **Modèle de classification**
classifier = RandomForestClassifier(random_state=42, class_weight='balanced')

```

```

# ❤ **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
    'classifier__class_weight': ['balanced', None] # Tester 'balanced' et None
}

# ❤ **Lancer la GridSearchCV avec le scorér personnalisé**
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring=custom_f1_scorer, verbose=2, n_jobs=-1)
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.unique(y), yticklabels=np.unique(y))
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.title("Matrice de Confusion")
plt.show()

```

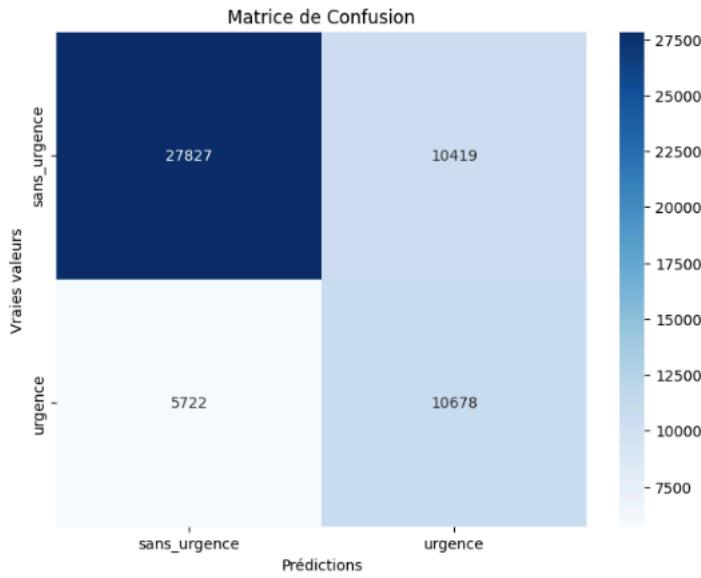
```

Meilleurs paramètres: {'classifier__class_weight': 'balanced', 'classifier__max_depth': 10, 'classifier__n_estimators': 50}

📊 **Performance du modèle optimal**
Accuracy: 0.70
Precision: 0.70
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     0.70     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)

#### h. 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', 'col', 'tué', 'blessé_léger', 'indemne', 'bleu'])
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 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 avec la régression logistique
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000, random_state=42))]) # Régression Logistique avec itérations supplémentaires

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

# Prédictions
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=['sans_urgence','urgence'], yticklabels=['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            -         -         -      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', 'col', 'tué', 'blessé_léger','indemne', 'ble
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', 'col', 'tué', 'blessé_léger','indemne', 'ble
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_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 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 avec RandomUnderSampler (undersampling) et la régression logistique
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling avec RandomUnderSampler
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression logistique
])

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

# Prédictions
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=['blessé_léger','blessé_hospitalisé', 'tué'], y
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_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 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 avec SMOTE (oversampling) et la régression logistique
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('smote', SMOTE(random_state=42)), # Oversampling avec SMOTE
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression logistique
])

# Entrainer 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=['blessé_léger','blessé_hospitalisé', 'tué'], y
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', 'col', 'tué', 'blessé_léger','indemne', 'ble
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', 'col', 'tué', 'blessé_léger','indemne', 'ble
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_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 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 avec RandomUnderSampler (undersampling) et la régression logistique
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling avec RandomUnderSampler
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression logistique
])

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

# Prédictions
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=['blessé_léger','blessé_hospitalisé', 'tué'], y
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_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 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 avec RandomUnderSampler (undersampling) et la régression logistique
pipeline = imPipeline([
    ('preprocessor', preprocessor), # Prétraitement des données
    ('undersample', RandomUnderSampler(random_state=42)), # Undersampling avec RandomUnderSampler
    ('classifier', LogisticRegression(max_iter=1000, random_state=42)) # Régression logistique
])

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

# Prédictions
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=['blessé_léger','blessé_hospitalisé', 'tué'], y
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     0.68     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

### i. 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 permet d'avoir un fichier mis à jour si modifi

```

```

df=pd.read_csv('/content/drive/MyDrive/Datascientest/Projet_accidents/Dataset/2019_2023/df_machine_learning.csv')
df['gravité_accident'] = df['gravité_accident'].replace({2: 'sans_urgence', 3: 'urgence', 4: 'sans_urgence'})
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_score, confusion_matrix, classification_report
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', 'col', 'tué', 'blessé_léger', 'indemne', 'blé'])
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', 'autoroute', 'autre_route',
'sens_unique', 'bidirectionnel', 'route_seche',
'route_mouillée_enneigée', '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 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 avec KNN
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier(n_neighbors=5))] ) # KNN avec 5 voisins

# Créer le scoré pour optimiser le recall
recall_scorer = make_scorer(recall_score, average='weighted')

# Recherche des hyperparamètres avec GridSearchCV
param_grid = {'classifier__n_neighbors': [3, 5, 7, 9, 11]} # Exemple de grid de voisins

# GridSearchCV pour optimiser le recall
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring=recall_scorer)
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}")

# Prédictions 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=['sans_urgence', 'urgence'], yticklabels=['sans_urgence', 'urgence'])
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     0.70     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

## j. 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, LabelEncoder
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_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

# Séparation des variables d'entrée et de sortie
X = df.drop(['lat', 'long', 'jour', 'mois', 'date', 'an', 'hrmn', 'dep', 'int', 'col', 'tué', 'blessé_léger',
             '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 0 et 1

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

# Variables catégorielles et cycliques

```

```

categorical_features = ['agg', 'atm'] # Variables à encoder
cyclical_features = ['heure'] # Transformation cyclique
passthrough_features = ['nbv', 'vma', 'nationale_departementale_communale', 'autoroute', 'autre_route',
                        '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 inchangées

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

# 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 multiclasses
    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)

# Prédictions
y_pred = model.predict(X_test_transformed)

# Évaluation du modèle
accuracy = accuracy_score(y_test, y_pred) # Les labels sont déjà sous forme numérique
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_score, confusion_matrix, classification_report
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', 'col', 'tué', 'blessé_léger', 'indemne', 'bles'])
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_urgence' → 0 et 'urgence' → 1

# 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', 'autoroute', 'autre_route',
'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 inchangées

# 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_metric='mlogloss', random_state=42)) # XGBoost
])

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

# Prédictions
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=['Classe 0', 'Classe 1'], yticklabels=['Classe 0', 'Classe 1'])
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['classifier']

# 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 encodées
# et la transformation cyclique pour obtenir les noms des features résultants.
cat_columns = pipeline.named_steps['preprocessor'].transformers_[0][1].get_feature_names_out(categorical_features)
cyclical_columns = ['heure_sin', 'heure_cos'] # Ces noms sont définis par la transformation de l'heure

# 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 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
      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.**

## Annexe: Description des bases de données annuelles

Observatoire national interministériel de la sécurité routière

Le 23 octobre 2024

Description des bases de données annuelles des accidents corporels de la circulation routière

Années de 2005 à 2023

Pour chaque accident corporel (soit un accident survenu sur une voie ouverte à la circulation publique, impliquant au moins un véhicule et ayant fait au moins une victime ayant nécessité des soins), des saisies d'information décrivant l'accident sont effectuées par l'unité des forces de l'ordre (police, gendarmerie, etc.) qui est intervenue sur le lieu de l'accident. Ces saisies sont rassemblées dans une fiche intitulée bulletin d'analyse des accidents corporels. L'ensemble de ces fiches constitue le fichier national des accidents corporels de la circulation dit « Fichier BAAC » administré par l'Observatoire national interministériel de la sécurité routière "ONISR".

Les bases de données, extraites du fichier BAAC, répertorient l'intégralité des accidents corporels de la circulation, intervenus durant une année précise en France métropolitaine, dans les départements d'Outre-mer (Guadeloupe, Guyane, Martinique, La Réunion et Mayotte depuis 2012) et dans les autres territoires d'outre-mer (Saint-Pierre-et-Miquelon, Saint-Barthélemy, Saint-Martin, Wallis-et-Futuna, Polynésie française et Nouvelle-Calédonie ; disponible qu'à partir de 2019 dans l'open data) avec une description simplifiée. Cela comprend des informations de localisation de l'accident, telles que renseignées ainsi que des informations concernant les caractéristiques de l'accident et son lieu, les véhicules impliqués et leurs victimes.

Par rapport aux bases de données agrégées 2005-2010 et 2006-2011 actuellement disponibles sur le site [www.data.gouv.fr](http://www.data.gouv.fr), les bases de données de 2005 à 2023 sont désormais annuelles et composées de 4 fichiers (Caractéristiques – Lieux – Véhicules – Usagers) au format csv.

Ces bases occultent néanmoins certaines données spécifiques relatives aux usagers et aux véhicules et à leur comportement dans la mesure où la divulgation de ces données porterait atteinte à la protection de la vie privée des personnes physiques aisément identifiables ou ferait apparaître le comportement de telles personnes alors que la divulgation de ce comportement pourrait leur porter préjudice (avis de la CADA – 2 janvier 2012).

Avertissement : Les données sur la qualification de blessé hospitalisé depuis l'année 2018 ne peuvent être comparées aux années précédentes suite à des modifications de processus de saisie des forces de l'ordre. L'indicateur « blessé hospitalisé » n'est plus labellisé par l'autorité de la statistique publique depuis 2019.

A partir des données de 2021, les usagers en fuite ont été rajoutés, cela entraîne des manques d'informations sur ces derniers, notamment le sexe, l'âge, voire la gravité des blessures (indemne, blessé léger ou blessé hospitalisé).

La validité des exploitations statistiques qui peuvent être faites à partir de cette base dépend des modes de vérifications propres dans le domaine d'application de la sécurité routière et notamment d'une connaissance précise des définitions différentes à chaque variable utilisée. Pour toute exploitation, il est important de prendre notamment connaissance de la structure de la fiche BAAC jointe ainsi que du guide d'utilisation de la codification du bulletin d'analyse des accidents corporels de la circulation.

Rappelons qu'un certain nombre d'indicateurs issus de cette base font l'objet d'une labellisation, par l'autorité de la statistique publique (arrêté du 27 novembre 2019).

La liste est disponible à l'adresse :

<https://www.onISR.securite-routiere.gouv.fr/outils-statistiques/indicateurs-labellises>

2

Définitions du fichier national des données BAAC

Bulletins d'Analyse des Accidents Corporels de la circulation

Un accident corporel (mortel et non mortel) de la circulation routière relevé par les forces de l'ordre :

- implique au moins une victime,
- survient sur une voie publique ou privée, ouverte à la circulation publique,
- implique au moins un véhicule.

Un accident corporel implique un certain nombre d'usagers. Parmi ceux-ci, on distingue :

- les personnes indemnes : impliquées non décédées et dont l'état ne nécessite aucun soin médical du fait de l'accident,
  - les victimes : impliquées non indemnes.
- o les personnes tuées : personnes qui décèdent du fait de l'accident, sur le coup ou dans les trente jours qui suivent l'accident,
  - o les personnes blessées : victimes non tuées.

- les blessés dits « hospitalisés » : victimes hospitalisées plus de 24 heures,
- les blessés légers : victimes ayant fait l'objet de soins médicaux mais n'ayant pas été admises comme patients à l'hôpital plus de 24 heures.

D'après la loi du 9 août 2004 relative à la politique de santé publique et l'arrêté du 27 mars 2007.

Définitions conformes à la décision du Conseil de l'Union européenne 93/704/CE du 30 novembre 1993 créant la base statistique européenne en matière d'accidentalité (dénommée « CARE» pour Community road accident database) et précisant les obligations des Etats membres en matière de transmission de statistiques d'accidentalité routière.

L'instruction ministérielle INTS17111J du 18 avril 2017 a diffusé le guide technique de rédaction des BAAC. L'instruction et le guide sont téléchargeables à l'adresse suivante :

<https://www.onisr.securite-routiere.gouv.fr/outils-statistiques/methodologies-statistiques>

#### Spécifications de la base

La base Etabl de données des accidents corporels de la circulation d'une année donnée, est répartie en 4 rubriques sous la forme pour chacune d'elles d'un fichier au format csv.

1. La rubrique CARACTERISTIQUES qui décrit les circonstances générales de l'accident
2. La rubrique LIEUX qui décrit le lieu principal de l'accident même si celui-ci s'est déroulé à une intersection
3. La rubrique VEHICULES impliqués
4. La rubrique USAGERS impliqués

Chacune des variables contenues dans une rubrique doit pouvoir être reliée aux variables des autres rubriques. Le n° d'identifiant de l'accident (Cf. "Num\_Acc") présent dans ces 4 rubriques permet d'établir un lien entre toutes les variables qui décrivent un accident. Quand un accident comporte plusieurs véhicules, il faut aussi pouvoir relier chaque véhicule à ses occupants. Ce lien est fait par la variable id\_vehicule.

La plupart des variables contenues dans les quatre fichiers précédemment énumérés peuvent contenir des cellules vides ou un zéro ou un point. Il s'agit, dans ces trois cas, d'une cellule non renseignée par les forces de l'ordre ou sans objet.

Un accident peut être géolocalisé de plusieurs façons :

- adresse partielle non normalisée (champ adr)
- coordonnées gps (projection WGS84)
- numéro de la route, PR de rattachement et distance curviline à ce PR

Attention : les accidents ne sont pas tous géolocalisés de façon précise à travers les informations disponibles dans le Fichier BAAC et restituées ici. A minima, seule la commune de l'accident est fournie.

Il s'agit d'une base brute non corrigée des erreurs de saisies qui font l'objet d'un processus de corrections ultérieurement. Les utilisateurs de la présente base sont invités à nous faire part, par courriel, de toutes anomalies qu'ils auraient relevées au cours de son exploitation.

4

Liste complète des champs avec le détail de leur contenu pour chaque fichier

En 2019, la base de données des accidents a évolué, dans le descriptif ci-dessous, en vert les nouvelles modalités de certaines variables et les nouvelles variables ajoutées.

#### La rubrique CARACTERISTIQUES

##### Num\_Acc

Numéro d'identifiant de l'accident.

jour

Jour de l'accident.

mois

Mois de l'accident.

an

Année de l'accident.

hrmn

Heure et minutes de l'accident.

lum

Lumière : conditions d'éclairage dans lesquelles l'accident s'est produit :

- 1 – Plein jour
- 2 – Crémuscle ou aube
- 3 – Nuit sans éclairage public
- 4 – Nuit avec éclairage public non allumé
- 5 – Nuit avec éclairage public allumé

dep

Département : Code INSEE (Institut National de la Statistique et des Etudes Economiques) du département (2A Corse-du-Sud – 2B Haute-Corse).

com

Commune : Le numéro de commune est un code donné par l'INSEE. Le code est composé du code INSEE du département suivi par 3 chiffres.

agg

Localisation :

- 1 – Hors agglomération 2 – En agglomération

int

Intersection :

- 1 – Hors intersection
- 2 – Intersection en X
- 3 – Intersection en T
- 4 – Intersection en Y
- 5 – Intersection à plus de 4 branches
- 6 – Giratoire
- 7 – Place
- 8 – Passage à niveau
- 9 – Autre intersection

5

atm

Conditions atmosphériques :

- 1 – Non renseigné
- 1 – Normale
- 2 – Pluie légère
- 3 – Pluie forte
- 4 – Neige - grêle
- 5 – Brouillard - fumée
- 6 – Vent fort - tempête
- 7 – Temps éblouissant
- 8 – Temps couvert
- 9 – Autre

col

Type de collision :

- 1 – Non renseigné
- 1 – Deux véhicules - frontale
- 2 – Deux véhicules – par l'arrière

- 3 – Deux véhicules – par le coté
- 4 – Trois véhicules et plus – en chaîne
- 5 – Trois véhicules et plus - collisions multiples
- 6 – Autre collision
- 7 – Sans collision

adr

Adresse postale : variable renseignée pour les accidents survenus en agglomération.

lat

Latitude

long

Longitude

La rubrique LIEUX

Num\_Acc

Identifiant de l'accident identique à celui du fichier "rubrique CARACTERISTIQUES" repris dans l'accident.

catr

Catégorie de route :

- 1 – Autoroute
- 2 – Route nationale
- 3 – Route Départementale
- 4 – Voie Communales
- 5 – Hors réseau public
- 6 – Parc de stationnement ouvert à la circulation publique
- 7 – Routes de métropole urbaine
- 9 – autre

voie

Numéro de la route.

V1

Indice numérique du numéro de route (exemple : 2 bis, 3 ter etc.).

6

V2

Lettre indice alphanumérique de la route.

circ

Régime de circulation :

- 1 – Non renseigné
- 1 – A sens unique
- 2 – Bidirectionnelle
- 3 – A chaussées séparées
- 4 – Avec voies d'affectation variable

nbv

Nombre total de voies de circulation.

vosp

Signale l'existence d'une voie réservée, indépendamment du fait que l'accident ait lieu ou non sur cette voie.

- 1 – Non renseigné

- 0 – Sans objet

- 1 – Piste cyclable
- 2 – Bande cyclable
- 3 – Voie réservée

prof

Profil en long décrit la déclivité de la route à l'endroit de l'accident :

- 1 – Non renseigné
- 1 – Plat
- 2 – Pente
- 3 – Sommet de côte
- 4 – Bas de côte

pr

Numéro du PR de rattachement (numéro de la borne amont). La valeur -1 signifie que le PR n'est pas renseigné.

pr1

Distance en mètres au PR (par rapport à la borne amont). La valeur -1 signifie que le PR n'est pas renseigné.

plan

Tracé en plan :

- 1 – Non renseigné
- 1 – Partie rectiligne
- 2 – En courbe à gauche
- 3 – En courbe à droite
- 4 – En « S »

lartpc

Largeur du terre-plein central (TPC) s'il existe (en m).

larrout

Largeur de la chaussée affectée à la circulation des véhicules ne sont pas compris les bandes d'arrêt d'urgence, les TPC et les places de stationnement (en m).

surf

Etat de la surface :

- 1 – Non renseigné
- 1 – Normale
- 2 – Mouillée
- 3 – Flaques
- 4 – Inondée
- 5 – Enneigée
- 6 – Boue
- 7 – Verglacée
- 8 – Corps gras – huile
- 9 – Autre

infra

Aménagement - Infrastructure :

- 1 – Non renseigné
- 0 – Aucun
- 1 – Souterrain - tunnel
- 2 – Pont - autopont
- 3 – Bretelle d'échangeur ou de raccordement
- 4 – Voie ferrée
- 5 – Carrefour aménagé

- 6 – Zone piétonne
- 7 – Zone de péage
- 8 – Chantier
- 9 – Autres

situ

Situation de l'accident :

- 1 – Non renseigné
- 0 – Aucun
- 1 – Sur chaussée
- 2 – Sur bande d'arrêt d'urgence
- 3 – Sur accotement
- 4 – Sur trottoir
- 5 – Sur piste cyclable
- 6 – Sur autre voie spéciale
- 8 – Autres

vma

Vitesse maximale autorisée sur le lieu et au moment de l'accident.

#### La rubrique VÉHICULES

Num\_Acc

Identifiant de l'accident identique à celui du fichier "rubrique CARACTERISTIQUES" repris pour chacun des véhicules décrits impliqués dans l'accident.

id\_vehicule

Identifiant unique du véhicule repris pour chacun des usagers occupant ce véhicule (y compris les piétons qui sont rattachés aux véhicules qui les ont heurtés) – Code numérique.

Num\_Veh

Identifiant du véhicule repris pour chacun des usagers occupant ce véhicule (y compris les piétons qui sont rattachés aux véhicules qui les ont heurtés) – Code alphanumérique.

senc

Sens de circulation :

- 1 – Non renseigné
- 8
- 0 – Inconnu
- 1 – PK ou PR ou numéro d'adresse postale croissant
- 2 – PK ou PR ou numéro d'adresse postale décroissant
- 3 – Absence de repère

catv

Catégorie du véhicule :

- 00 – Indéterminable
- 01 – Bicyclette
- 02 – Cyclomoteur <50cm<sup>3</sup>
- 03 – Voiturette (Quadricycle à moteur carrossé) (anciennement "voiturette ou tricycle à moteur")
- 04 – Référence inutilisée depuis 2006 (scooter immatriculé)
- 05 – Référence inutilisée depuis 2006 (motocyclette)
- 06 – Référence inutilisée depuis 2006 (side-car)
- 07 – VL seul
- 08 – Référence inutilisée depuis 2006 (VL + caravane)
- 09 – Référence inutilisée depuis 2006 (VL + remorque)

- 10 – VU seul 1,5T <= PTAC <= 3,5T avec ou sans remorque (anciennement VU seul 1,5T <= PTAC <= 3,5T)  
 11 – Référence inutilisée depuis 2006 (VU (10) + caravane)  
 12 – Référence inutilisée depuis 2006 (VU (10) + remorque)  
 13 – PL seul 3,5T < PTCA <= 7,5T  
 14 – PL seul > 7,5T  
 15 – PL > 3,5T + remorque  
 16 – Tracteur routier seul  
 17 – Tracteur routier + semi-remorque  
 18 – Référence inutilisée depuis 2006 (transport en commun)  
 19 – Référence inutilisée depuis 2006 (tramway)  
 20 – Engin spécial  
 21 – Tracteur agricole  
 30 – Scooter < 50 cm<sup>3</sup>  
 31 – Motocyclette > 50 cm<sup>3</sup> et <= 125 cm<sup>3</sup>  
 32 – Scooter > 50 cm<sup>3</sup> et <= 125 cm<sup>3</sup>  
 33 – Motocyclette > 125 cm<sup>3</sup>  
 34 – Scooter > 125 cm<sup>3</sup>  
 35 – Quad léger <= 50 cm<sup>3</sup> (Quadricycle à moteur non carrossé)  
 36 – Quad lourd > 50 cm<sup>3</sup> (Quadricycle à moteur non carrossé)  
 37 – Autobus  
 38 – Autocar  
 39 – Train  
 40 – Tramway  
 41 – 3RM <= 50 cm<sup>3</sup>  
 42 – 3RM > 50 cm<sup>3</sup> <= 125 cm<sup>3</sup>  
 43 – 3RM > 125 cm<sup>3</sup>  
 50 – EDP à moteur  
 60 – EDP sans moteur  
 80 – VAE  
 99 – Autre véhicule

obs

- Obstacle fixe heurté :
- 1 – Non renseigné
  - 0 – Sans objet
  - 1 – Véhicule en stationnement
  - 2 – Arbre
  - 3 – Glissière métallique
  - 4 – Glissière béton
  - 5 – Autre glissière
  - 6 – Bâtiment, mur, pile de pont
  - 7 – Support de signalisation verticale ou poste d'appel d'urgence
  - 8 – Poteau
  - 9 – Mobilier urbain
  - 10 – Parapet
  - 11 – Ilot, refuge, borne haute
  - 12 – Bordure de trottoir
  - 13 – Fossé, talus, paroi rocheuse
  - 14 – Autre obstacle fixe sur chaussée
  - 15 – Autre obstacle fixe sur trottoir ou accotement
  - 16 – Sortie de chaussée sans obstacle
  - 17 – Buse – tête d'aqueduc

obsm

- Obstacle mobile heurté :
- 1 – Non renseigné
  - 0 – Aucun

- 1 – Piéton
- 2 – Véhicule
- 4 – Véhicule sur rail
- 5 – Animal domestique
- 6 – Animal sauvage
- 9 – Autre

choc

Point de choc initial :

- 1 – Non renseigné
- 0 – Aucun
- 1 – Avant
- 2 – Avant droit
- 3 – Avant gauche
- 4 – Arrière
- 5 – Arrière droit
- 6 – Arrière gauche
- 7 – Côté droit
- 8 – Côté gauche
- 9 – Chocs multiples (tonneaux)

manv

Manoeuvre principale avant l'accident :

- 1 – Non renseigné
- 0 – Inconnue
- 1 – Sans changement de direction
- 2 – Même sens, même file
- 3 – Entre 2 files
- 4 – En marche arrière
- 5 – A contresens
- 6 – En franchissant le terre-plein central
- 7 – Dans le couloir bus, dans le même sens
- 8 – Dans le couloir bus, dans le sens inverse
- 9 – En s'insérant
- 10 – En faisant demi-tour sur la chaussée
- Changeant de file
- 11 – A gauche
- 12 – A droite
- 10
- Déporté
- 13 – A gauche
- 14 – A droite
- Tournant
- 15 – A gauche
- 16 – A droite
- Dépassant
- 17 – A gauche
- 18 – A droite
- Divers
- 19 – Traversant la chaussée
- 20 – Manoeuvre de stationnement
- 21 – Manoeuvre d'évitement
- 22 – Ouverture de porte
- 23 – Arrêté (hors stationnement)
- 24 – En stationnement (avec occupants
- 25 – Circulant sur trottoir
- 26 – Autres manoeuvres

motor

Type de motorisation du véhicule :

-1 – Non renseigné

0 – Inconnue

1 – Hydrocarbures

2 – Hybride électrique

3 – Electrique

4 – Hydrogène

5 – Humaine

6 – Autre

occutc

Nombre d'occupants dans le transport en commun.

La rubrique USAGERS

Num\_Acc

Identifiant de l'accident identique à celui du fichier "rubrique CARACTERISTIQUES" repris pour chacun des usagers décrits impliqués dans l'accident.

id\_usager

Identifiant unique de l'usager (y compris les piétons qui sont rattachés aux véhicules qui les ont heurtés) – Code numérique.

id\_vehicule

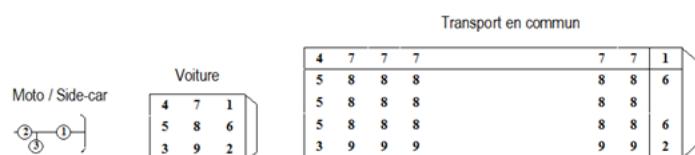
Identifiant unique du véhicule repris pour chacun des usagers occupant ce véhicule (y compris les piétons qui sont rattachés aux véhicules qui les ont heurtés) – Code numérique.

num\_Veh

Identifiant du véhicule repris pour chacun des usagers occupant ce véhicule (y compris les piétons qui sont rattachés aux véhicules qui les ont heurtés) – Code alphanumérique.

place

Permet de situer la place occupée dans le véhicule par l'usager au moment de l'accident. Le détail est donné par l'illustration ci-dessous :



10 – Piéton (non applicable)

catu

Catégorie d'usager :

1 – Conducteur

2 – Passager

3 – Piéton

grav

Gravité de blessure de l'usager, les usagers accidentés sont classés en trois catégories de victimes plus les indemnes :

- 1 – Indemne
- 2 – Tué
- 3 – Blessé hospitalisé
- 4 – Blessé léger

sex

Sexe de l'usager :

- 1 – Masculin
- 2 – Féminin

An\_nais

Année de naissance de l'usager.

trajet

Motif du déplacement au moment de l'accident :

- 1 – Non renseigné
- 0 – Non renseigné
- 1 – Domicile – travail
- 2 – Domicile – école
- 3 – Courses – achats
- 4 – Utilisation professionnelle
- 5 – Promenade – loisirs
- 9 – Autre

Les équipements de sécurité jusqu'en 2018 étaient en 2 variables : existence et utilisation.

A partir de 2019, il s'agit de l'utilisation avec jusqu'à 3 équipements possibles pour un même usager (notamment pour les motocyclistes dont le port du casque et des gants est obligatoire).

secu1

Le renseignement du caractère indique la présence et l'utilisation de l'équipement de sécurité :

- 1 – Non renseigné
- 0 – Aucun équipement
- 1 – Ceinture
- 2 – Casque
- 3 – Dispositif enfants
- 4 – Gilet réfléchissant
- 5 – Airbag (2RM/3RM)
- 6 – Gants (2RM/3RM)
- 7 – Gants + Airbag (2RM/3RM)
- 8 – Non déterminable
- 9 – Autre

secu2

Le renseignement du caractère indique la présence et l'utilisation de l'équipement de sécurité :

- 1 – Non renseigné
- 0 – Aucun équipement
- 1 – Ceinture
- 2 – Casque
- 3 – Dispositif enfants
- 4 – Gilet réfléchissant
- 5 – Airbag (2RM/3RM)
- 6 – Gants (2RM/3RM)
- 7 – Gants + Airbag (2RM/3RM)
- 8 – Non déterminable
- 9 – Autre

secu3

Le renseignement du caractère indique la présence et l'utilisation de l'équipement de sécurité :

- 1 – Non renseigné
- 0 – Aucun équipement
- 1 – Ceinture
- 2 – Casque
- 3 – Dispositif enfants
- 4 – Gilet réfléchissant
- 5 – Airbag (2RM/3RM)
- 6 – Gants (2RM/3RM)
- 7 – Gants + Airbag (2RM/3RM)
- 8 – Non déterminable
- 9 – Autre

locp

Localisation du piéton :

- 1 – Non renseigné
- 0 – Sans objet
- Sur chaussée :
  - 1 – A + 50 m du passage piéton
  - 2 – A – 50 m du passage piéton
- Sur passage piéton :
  - 3 – Sans signalisation lumineuse
  - 4 – Avec signalisation lumineuse
- Divers :
  - 5 – Sur trottoir
  - 6 – Sur accotement
  - 7 – Sur refuge ou BAU
  - 8 – Sur contre allée
  - 9 – Inconnue

actp

Action du piéton :

- 1 – Non renseigné
- 13
- Se déplaçant
  - 0 – Non renseigné ou sans objet
  - 1 – Sens véhicule heurtant
  - 2 – Sens inverse du véhicule
- Divers
  - 3 – Traversant
  - 4 – Masqué
  - 5 – Jouant – courant
  - 6 – Avec animal
  - 9 – Autre
- A – Monte/descend du véhicule
- B – Inconnue

etatp

Cette variable permet de préciser si le piéton accidenté était seul ou non :

- 1 – Non renseigné
- 1 – Seul
- 2 – Accompagné
- 3 – En groupe

Jointure des tables

