Parhelion Group

Analyse prédictive des sinistres automobiles corporels

Comréhension métien

En 2017, 3 448 personnes ont perdu la vie dans un accident de la route en France métropolitaine. Avec 29 décès de moins, la mortalité routière est en légère baisse (-0,8%) par rapport à 2016, après deux années d'augmentation, en 2014 (+3,5%) et en 2015 (+2,3%) et une stabilisation en 2016 (+0,46%).</Br>
Les autres indicateurs de l'accidentalité sont en légère hausse : le nombre de personnes blessées sur les routes augmente de +1,0%, soit au total 73 384 personnes blessées dans les 58 613 accidents corporels (+1,9%). 27 732 de ces personnes ont dû être hospitalisées (+2,0% par rapport à 2016) parmi lesquelles une sur dix gardera des séquelles lourdes.
Mais on doit se poser les questions suivantes : Dans quels cas on a des accidents graves ? Les accidents graves sont faits par les femmes ou les hommes ? Quels sont leurs âges, leurs catégories ?</br>

Si un accident auto est déclaré, et le gestionnaire a besoin d'estimer la gravité de l'accident. A-t-il besoin ainsi d'un algorithme pour lui aider.</br>

En effet,pour attribuer un tarif à ses clients , une assurance a besoin d'estimer le nombre d'accidents qu'il pourrait y avoir lieu en prenant en compte plusieurs criteres :</br>

+La durée d'un contrat d'assurance automobile est généralement d'un an, renouvelable par tacite reconduction ==> on peut donc avoir recours à une une alalyse des données sur une année par commune ou departement</br>

+le type de vehicule est un facteur à prendre en consideration : on peut diviser nos données en tranches d'age</br>

+age du conducteur</br>

+utilisation du vehicule </br>

+dispositif de securité</br>

+type du vehicule</br>

+sexe de l'utilisateur</br>

+lieu de residence (Pour cela on va se baser sur la commune où reside le conducteur)</br>

+heures d'utilisation du véhicule les plus fréquente (aube , matin , soir)</br>

</br> **Objectif Metier**</br>

Indice sur lesquels on va travailler :</br>

_Probabilité d'avoir un accident pour une année choisie en prenant en consideration tout les criteres mentionnés ci-dessus.</br>

_Probabilité d'avoir un accident grave (avec mort)</br>

_Gravité des accidents</br>

Pour estimer la valeur des dégats en cas d'accident , et en prenant compte des critére mentionnés ci dessus on peut par exemple calculer la probabilité que le choc initial soit :</br>

_à l'arriere _à l'avant _sur les cotés ou multiple</br>

Pour savoir le nombre d'ambulancier qu'un hôpital doit mobiliser on pourrait s'amuser à calculer le nombre d'accident qui peuvent avoir lieu durant la nuit / jour </br>
On va aussi travailler sur la segmentation des zones géographiques

_Segmentation les zones géographiques

_Déterminer les conditions environnementales des communes

In [1]:

```
import numpy as np
import pandas as pd
```

```
from scipy import stats
```

In [2]:

```
lieux=pd.DataFrame()
for i in range(2005,2017):
    l=pd.io.parsers.read_csv(r"lieux-"+str(i)+".csv")
    l["Annee"]=i
    lieux=pd.concat([lieux,l])
```

```
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2785:
DtypeWarning: Columns (2) have mixed types. Specify dtype option on import or set
low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [3]:

```
vehicules = pd.io.parsers.read_csv(r"vehicules-2017.csv",sep=",");
caract = pd.io.parsers.read_csv(r"caracteristiques-2017.csv",sep=",",encoding="latin1");
lieux = pd.io.parsers.read_csv(r"lieux-2017.csv",sep=",",low_memory=False);
usagers =pd.io.parsers.read_csv(r"usagers-2017.csv",sep=",");
#for i in range(2005,2016):
#    vehicules = vehicules+ pd.io.parsers.read_csv(r"vehicules-"+str(i)+".csv",sep=",");
#    if(i!=2009):
 #      caract =caract+ pd.io.parsers.read_csv(r"caracteristiques-"+str(i)+".csv",encoding = "ISO-
8859-1",sep=",");
  #  else:
  #      caract =caract+ pd.io.parsers.read_csv(r"caracteristiques-"+str(i)+".csv",encoding =
"ISO-8859-1",sep="\t");
   # usagers = usagers+pd.io.parsers.read_csv(r"usagers-"+str(i)+".csv",sep=",");
```

In [4]:

```
#joindre les données dans une seule table , clé de jointure : Numéro de l'accident
df= pd.merge(left=vehicules,right=caract,on="Num_Acc")
df= pd.merge(left=df,right=lieux,on="Num_Acc")
df= pd.merge(left=df,right=usagers,on=["Num_Acc","num_veh"])
df.shape
```

Out[4]:

```
(136021, 51)
```

Visualisation des données

Ces dataViz sont réalisés par tableau

Gravité des accidents

Comparer les niveaux de gravité des accidents et le nombre des accidents en fonction de quelques paramètres

1.Sexe

L'idée est d'avoir une comparaison visuelle entre le nombre des accidents réalisés par les femmes et les hommes en tenant compte de la gravité

Conclusion: On peut voir que le nombre des accidents mortels est très faible pour les femmes ainsi que les hommes par rapport aux autres gravités.</br> On peut voir également que le grand nombre des accidents correspond au gravité "indemne" mais pour les femmes la gravité fréquente est Blessé Leger

2.Catégorie d’usagers

On veut voir le nombre d'accident en prenant en compte la catégorie des usagers, le sexe et la gravité

Conclusion On remarque que la modalitée la plus fréquente en nombre des accidents est conducteur pour les hommes et les femmes sachant tout les gravités.

3. Ages

On va voir la gravité des accidents en fonction de l’âge et le nombre des accidents

Conclusion En voyant toutes les gravités on remarque que le nombre d'accidents est élevé chez les personnes entres 19 et 24

Préparation des données

In [5]:

```
#ne garder que les données qui concernent le conducteur
df=df.loc[df.catu==1,]
df.shape
```

Out[5]:

```
(100534, 51)
```

In [6]:

```
df.secu.head()
```

Out[6]:

```
0    13.0
2    13.0
3    11.0
4    22.0
5    11.0
Name: secu, dtype: float64
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['Num_Acc', 'senc', 'catv', 'occutc', 'obs', 'obsm', 'choc', 'manv',
       'num_veh', 'an', 'mois', 'jour', 'hrmn', 'lum', 'agg', 'int', 'atm',
       'col', 'com', 'adr', 'gps', 'lat', 'long', 'dep', 'catr', 'voie', 'v1',
       'v2', 'circ', 'nbv', 'pr', 'pr1', 'vosp', 'prof', 'plan', 'lartpc',
       'larrout', 'surf', 'infra', 'situ', 'env1', 'place', 'catu', 'grav',
       'sexe', 'trajet', 'secu', 'locp', 'actp', 'etatp', 'an_nais'],
      dtype='object')
```

In [8]:

```
#voir si nos données contiennent des NA
df.isna().sum()
```

Out[8]:

```
Num_Acc         0
senc           49
catv            0
occutc          0
obs            37
obsm           26
choc           16
manv           13
num_veh         0
an              0
mois            0
jour            0
hrmn            0
lum             0
agg             0
int             0
atm            17
col            10
com             0
adr          1360
gps          7535
lat         13295
long        13295
dep             0
catr            0
voie        15323
v1          99891
v2          95914
circ          631
nbv           742
pr          54185
pr1         54576
vosp         1050
prof          781
plan         1461
lartpc       3624
larrout      3402
surf          810
infra        6537
situ         6207
env1         6621
place           0
catu            0
grav            0
sexe            0
trajet          4
secu           27
locp           29
actp           30
etatp          48
an_nais        23
dtype: int64
```

In [9]:

```
#v1,v2,pr,pr1 ont + >50% de valeurs manquantes on va les négliger
df.drop(columns=["num_veh","v1","v2","pr","pr1"],inplace=True)
df.head()
```

Out[9]:

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp
0	201700000001	0.0	7	0	0.0	2.0	3.0	9.0	17	1	...	1.0	1	3	1	9.0	13.0	0.0	0.0
2	201700000001	0.0	10	0	0.0	2.0	3.0	13.0	17	1	...	1.0	1	3	1	1.0	13.0	0.0	0.0

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp
3	201700000002	0.0	7	0	0.0	0.0	1.0	16.0	17	2	...	1.0	1	1	1	0.0	11.0	0.0	0.0
4	201700000002	0.0	1	0	0.0	0.0	7.0	1.0	17	2	...	1.0	1	3	1	5.0	22.0	0.0	0.0
5	201700000003	0.0	10	0	0.0	2.0	1.0	1.0	17	3	...	1.0	1	1	1	1.0	11.0	0.0	0.0

5 rows × 46 columns

In [10]:

```
#on remplace les autres na par les valeurs du mode de chaque variable

df['senc'].fillna(df.senc.mode()[0],inplace=True)
df['obs'].fillna(df.obs.mode()[0],inplace=True)
df['obsm'].fillna(df.obsm.mode()[0],inplace=True)
df['choc'].fillna(df.choc.mode()[0],inplace=True)
df['manv'].fillna(df.manv.mode()[0],inplace=True)
df['atm'].fillna(df.atm.mode()[0],inplace=True)
df['col'].fillna(df.col.mode()[0],inplace=True)
df['adr'].fillna(df.adr.mode()[0],inplace=True)
df['gps'].fillna(df.gps.mode()[0],inplace=True)
df['lat'].fillna(df.lat.mode()[0],inplace=True)
df['long'].fillna(df.long.mode()[0],inplace=True)
df['voie'].fillna(df.voie.mode()[0],inplace=True)
df['circ'].fillna(df.circ.mode()[0],inplace=True)
df['nbv'].fillna(df.nbv.mode()[0],inplace=True)
df['vosp'].fillna(df.vosp.mode()[0],inplace=True)
df['prof'].fillna(df.prof.mode()[0],inplace=True)
df['plan'].fillna(df.plan.mode()[0],inplace=True)
df['lartpc'].fillna(df.lartpc.mode()[0],inplace=True)
df['larrout'].fillna(df.larrout.mode()[0],inplace=True)
df['surf'].fillna(df.surf.mode()[0],inplace=True)
df['infra'].fillna(df.infra.mode()[0],inplace=True)
df['situ'].fillna(df.situ.mode()[0],inplace=True)
df['env1'].fillna(df.env1.mode()[0],inplace=True)
df['trajet'].fillna(df.trajet.mode()[0],inplace=True)
df['secu'].fillna(df.secu.mode()[0],inplace=True)
df['locp'].fillna(df.locp.mode()[0],inplace=True)
df['actp'].fillna(df.actp.mode()[0],inplace=True)
df['etatp'].fillna(df.etatp.mode()[0],inplace=True)
df['an_nais'].fillna(df.an_nais.mode()[0],inplace=True)
```

In [11]:

```
#d'aprés la description des données , on peut voir que les départements sont suivis par un zero ,
on doit supprimer ce zero
df['dep']=df['dep']/10
```

In [12]:

```
#remplaceer l'année de naissance par l'age du conducteur
df.an_nais=df.an+2000-df.an_nais
```

In [13]:

```
df.loc[df.an_nais<=18,"an_nais"].value_counts().sort_index()
```

Out[13]:

```
2.0         1
3.0         1
5.0        10
6.0        11
7.0        14
8.0        21
9.0        27
10.0       30
11.0       62
12.0       56
13.0       96
14.0      235
15.0      556
16.0      807
```

```
17.0     1133
18.0     1437
Name: an_nais, dtype: int64
```

In [14]:

```
#Suppression des lignes où l age du conducteur est inferieur à 18 ans , vu q'il ne peuvent pas con
duire
df=df.loc[df.an_nais>=16,]
```

In [15]:

```
#décomposer l'age en tranches d'age [18-23] [24-35] [36-49] [50-69] [70+]
df.loc[(df.an_nais>=16)&(df.an_nais<=23),"an_nais"]=1
df.loc[(df.an_nais>=24)&(df.an_nais<=35),"an_nais"]=2
df.loc[(df.an_nais>=36)&(df.an_nais<=49),"an_nais"]=3
df.loc[(df.an_nais>=50)&(df.an_nais<=69),"an_nais"]=4
df.loc[(df.an_nais>=70),"an_nais"]=5
df.rename(columns={"an_nais":"age"},inplace=True)
```

In [16]:

```
#correction anomalies au niveau de secu:
df.loc[df.secu==40].secu
```

Out[16]:

```
Series([], Name: secu, dtype: float64)
```

In [17]:

```
df.loc[df.secu==31,"secu"]=df.secu.mode()[0]
```

In [18]:

```
#transformation de secu en : dispositif de securité utilisé ou non (binaire)
df.loc[df.secu%10==1,"secu"]=1
df.loc[df.secu%10!=1,"secu"]=0
```

In [19]:

```
#transformation de lum en trois catégories (utilisation la plus frequente à l aube , le jour , ou
bien la nuit ?)
df.loc[df.lum==2,"lum"]=1 #aube
df.loc[df.lum==1,"lum"]=2 #matin
df.loc[df.lum!=2,"lum"]=3 #soir
```

In [20]:

```
#elimination des moyens de transport commun (vu que notre etude concerne une assurance privée)
df=df[(df.catv!=18) & (df.catv!=19) & (df.catv!=37) & (df.catv!=38) & (df.catv!=39) & (df.catv!=40)
]
df.shape
```

Out[20]:

```
(98317, 46)
```

In [21]:

```
df2=pd.DataFrame(df)
df2.head()
```

Out[21]:

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp
0	201700000001	0.0	7	0	0.0	2.0	3.0	9.0	17	1	...	1.0	1	3	1	9.0	0.0	0.0	0.0
2	201700000001	0.0	10	0	0.0	2.0	3.0	13.0	17	1	...	1.0	1	3	1	1.0	0.0	0.0	0.0
3	201700000002	0.0	7	0	0.0	0.0	1.0	16.0	17	2	...	1.0	1	1	1	0.0	1.0	0.0	0.0
4	201700000002	0.0	1	0	0.0	0.0	7.0	1.0	17	2	...	1.0	1	3	1	5.0	0.0	0.0	0.0
5	201700000003	0.0	10	0	0.0	2.0	1.0	1.0	17	3	...	1.0	1	1	1	1.0	1.0	0.0	0.0

5 rows × 46 columns

In [59]:

```
#ne garder que les variables dont on a besoin pour nos indices à savoir :  An_nais (date de naissa
nce) , trajet (utilisation du vehicule) , secu (utilisation du dispositif de securité) , catv (cat
egorie du vehicule) , sexe , com (commune de residence) , lum (heures d'utilisation les plus freq.
) , an (annee)
df.drop(columns=["Num_Acc","agg","atm","int","col","adr","gps","lat","long","catr","voie","circ","v
osp","prof","plan","surf","infra","situ","env1","senc","mois","jour","nbv","obs","obsm","choc","man
v","occutc","place","catu","locp","actp","etatp","hrmn","lartpc","larrout"],inplace=True)
```

In [100]:

```
#groupper et agréger les données par an et par commune
df['nbacc']=1
df=df.groupby(['catv','an','lum','com','dep','sexe','trajet','secu','age','grav'],as_index=False).a
gg({"nbacc":"count"})
```

In [101]:

```
df=pd.get_dummies(df,columns=['catv','lum','sexe','trajet','secu','age','grav'])
```

In [102]:

```
#charger un jeu de données contenant de informations géograpphques et démographiques
population = pd.read_csv(r"populat.csv",sep=";")
population.head(2)
```

Out[102]:

	Code INSEE	Code Postal	Commune	Département	Région	Statut	Altitude Moyenne	Superficie	Population	geo_point_2d
0	32460	32720	VERGOIGNAN	GERS	MIDI-PYRENEES	Commune simple	126.0	1056.0	0.3	43.7235746425, -0.188266221507
1	51141	51240	LA CHAUSSEE-SUR-MARNE	MARNE	CHAMPAGNE-ARDENNE	Commune simple	130.0	2240.0	0.7	48.8433156105, 4.54286173009

In [103]:

```
#garder les colonnes qui concernent : la superficie de la region , poupulation , code commune et d
epartement (pour la jointure)
# et l altitude moyenne
col_list=['Code INSEE','Altitude Moyenne','Superficie','Population','Code Commune','Code
Département']
population=population[col_list]
```

In [104]:

```
#jointure des données par code commune et departement
population.rename(columns={"Code Commune":"com","Code Département":"dep","Code INSEE":"code_insee"
,"Altitude Moyenne":"altitude_moy"},inplace=True)
population.loc[population.dep=="2A"]=20.1
population.loc[population.dep=="2B"]=20.2
population.dep=population.dep.astype(float)
df=pd.merge(left=df,right=population,on=["com","dep"])
```

```
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py:969: UserWarning: Yo
u are merging on int and float columns where the float values are not equal to their int
representation
  'representation', UserWarning)
```

In [105]:

```
#chargement d'un jeu de donnée contenant des données socio démographiques
socioD=pd.read_excel(r"MDB-INSEE-V2.xls")
socioD.head(2)
```

```
WARNING *** OLE2 inconsistency: SSCS size is 0 but SSAT size is non-zero
```

Out[105]:

	CODGEO	Nb Pharmacies et parfumerie	Dynamique Entrepreneuriale	Dynamique Entrepreneuriale Service et Commerce	Synergie Médicale COMMUNE	Orientation Economique	Indice Fiscal Partiel	Score Fiscal	Indice Evasion Client	Sc Evas Cli
0	01001	0.0	57.0	23.0	114	Bassin Industriel	101.93878	59.04139	0.0	0.0
1	01002	0.0	45.0	4.0	143	Bassin Résidentiel	101.93878	59.04139	0.0	0.0

2 rows × 101 columns

In [106]:

```
#jointure de socioD avec notre jeu de données principale
ind_list=['CODGEO','Score Démographique','Score Ménages','Evolution Population','Nb Femme','Nb
Homme','Nb Mineurs','Nb Majeurs','Nb Etudiants','Reg Moyenne Salaires Horaires','Score Urbanité','
Nb Education, santé, action sociale','Score Croissance Population']
socioD=socioD[ind_list]
socioD.rename(columns={"CODGEO":"code_insee"},inplace=True)
df=pd.merge(left=df,right=socioD,on="code_insee")
```

Modélisation

Linear regression

In [107]:

```
#Calcul de l'indice : Probabilité que le profil de l'assuré ait un accident par rapport à tout les
accidents ayant eu lieu dans la meme commune
#pteNbAcc contiendra cet indice ==> elle représente donc notre variable cible
df["pteNbAcc"]=1
aux=[]
for row in df.iterrows():
    aux.append(df.loc[(df.dep==row[1]["dep"])&(df.com==row[1]["com"]),].nbacc.sum())
df.pteNbAcc=aux
df.pteNbAcc=df.nbacc/df.pteNbAcc
```

In [108]:

```
df.drop(columns=["nbacc"],inplace=True)
```

In [109]:

```
x=df.drop(columns=['pteNbAcc'])
y=df['pteNbAcc']
```

In [110]:

```
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [111]:

```
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

r = linear_model.LinearRegression()
r.fit(X_train,y_train)
print('Intercept: \n',r.intercept_)
print('Coefficients: \n',r.coef_)
```

```
Intercept: 
 0.1682770661439111
Coefficients: 
 [ 3.93597798e-15 -1.91662574e+09 -2.83917557e+11 -1.72998991e+09
 -1.52936004e+09 -5.65009707e+08 -4.01658658e+09 -1.90555003e+09
 -5.95558659e+08 -8.62403445e+08 -9.02878914e+08 -2.31640592e+08
 -9.10908023e+08 -2.72950485e+08 -4.75389743e+08 -1.20533698e+09
 -1.35924898e+09 -7.48852105e+08 -2.48473445e+09 -9.96933592e+08
 -1.72684023e+08 -3.72084834e+08 -5.14301544e+08 -2.13108152e+10
 -2.13108152e+10  7.31955600e+10  7.31955600e+10 -1.40561997e+11
 -1.36157746e+11 -4.39825470e+10 -6.16493846e+10 -1.10045489e+11
 -1.74377864e+11 -9.82825279e+10  2.36756928e+10  2.36756928e+10
  1.56380090e+10  1.78027214e+10  1.73809522e+10  1.68425006e+10
  1.06279983e+10  9.41438525e+10  3.46347678e+10  8.01143105e+10
  8.69409623e+10  2.83697775e+11  1.09386444e-02 -2.95753479e-02
  2.94647217e-02  2.60453796e+00  4.16336060e-02  4.39682007e-02
 -9.22957547e+11 -8.13673565e+11  1.01824064e+12  7.18911263e+11
  4.69436646e-02 -6.46209717e-03 -1.21444702e-01  2.36053467e-02
 -3.70750427e-02]
```

Regression logistique

In [105]:

```
from sklearn.linear_model import LogisticRegression
```

In [106]:

```
from scipy.stats import multinomial
```

In [22]:

```
round(df2)
df2.head()
```

Out[22]:

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp
0	201700000001	0.0	7	0	0.0	2.0	3.0	9.0	17	1	...	1.0	1	3	1	9.0	0.0	0.0	0.0
2	201700000001	0.0	10	0	0.0	2.0	3.0	13.0	17	1	...	1.0	1	3	1	1.0	0.0	0.0	0.0

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp
3	201700000002	0.0	7	0	0.0	0.0	1.0	16.0	17	2	...	1.0	1	1	1	0.0	1.0	0.0	0.0
4	201700000002	0.0	1	0	0.0	0.0	7.0	1.0	17	2	...	1.0	1	3	1	5.0	0.0	0.0	0.0
5	201700000003	0.0	10	0	0.0	2.0	1.0	1.0	17	3	...	1.0	1	1	1	1.0	1.0	0.0	0.0

5 rows × 46 columns

In [42]:

```
df3=pd.DataFrame(df2)
```

In [116]:

```
df2.drop(columns=["Num_Acc","senc","trajet","gps","adr","voie"],inplace=True)
```

In [109]:

```
x1=df2.drop(columns=['grav'])
y1=df2['grav']
```

In [182]:

```
df2=df2.astype('float')
```

In [110]:

```
from sklearn.cross_validation import train_test_split
X_train1,X_test1,y_train1,y_test1=train_test_split(x1,y1,test_size=0.25,random_state=0)
```

In [121]:

```
# import the class
from sklearn.linear_model import LogisticRegression
# instantiate the model (using the default parameters)
logreg = LogisticRegression(solver="saga")
# fit the model with data
logreg.fit(X_train1,y_train1)
```

```
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326:
ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326:
ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326:
ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326:
ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

Out[121]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='saga', tol=0.0001,
          verbose=0, warm_start=False)
```

In [122]:

```
y_pred1=logreg.predict(X_test1)
```

In [134]:

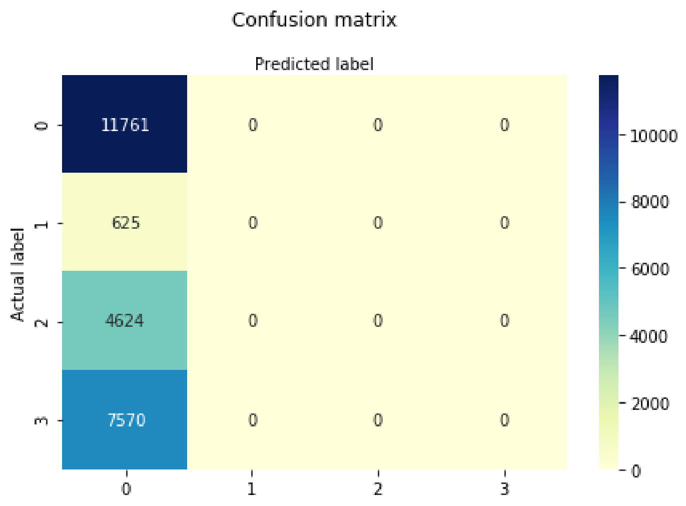
```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test1, y_pred1)
```

In [135]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[135]:

```
Text(0.5,257.44,'Predicted label')
```



In [219]:

```
logreg.score(x1,y1)
```

Out[219]:

```
0.47853372255052534
```

In [141]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test1,y_pred1))
```

```
             precision    recall  f1-score   support

          1       0.48      1.00      0.65     11761
          2       0.00      0.00      0.00       625
          3       0.00      0.00      0.00      4624
          4       0.00      0.00      0.00      7570

avg / total       0.23      0.48      0.31     24580


C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [ ]:

```python
print('Intercept: \n',logreg.intercept_)
print('Coefficients: \n',logreg.coef_)
```

Stepwise selection

In [111]:

```python
from sklearn.datasets import load_boston
import pandas as pd
import numpy as np
import statsmodels.api as sm
```

In [205]:

```python
def stepwise_selection(X, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):

    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.argmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included
```

In [206]:

```python
result = stepwise_selection(x1, y1)
print('resulting features:')
print(result)
```

```
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\statsmodels\base\model.py:1100: RuntimeWarning: i
nvalid value encountered in true_divide
  return self.params / self.bse
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:879:
RuntimeWarning: invalid value encountered in greater
  return (self.a < x) & (x < self.b)
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:879:
RuntimeWarning: invalid value encountered in less
  return (self.a < x) & (x < self.b)
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:1821:
RuntimeWarning: invalid value encountered in less_equal
  cond2 = cond0 & (x <= self.a)
C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\ipykernel_launcher.py:18: FutureWarning: 'argmin'
```

```
is deprecated, use 'idxmin' instead. The behavior of 'argmin'
will be corrected to return the positional minimum in the future.
Use 'series.values.argmin' to get the position of the minimum now.

Add  obs                            with p-value 0.0
Add  an                             with p-value 0.0
Add  catv                           with p-value 0.0
Add  catu                           with p-value 0.0
Add  sexe                           with p-value 1.3472e-182
Add  col                            with p-value 5.29512e-161
Add  age                            with p-value 5.63856e-150
Add  manv                           with p-value 4.89755e-148
Add  obsm                           with p-value 6.62087e-86
Add  agg                            with p-value 1.24024e-73
Add  secu                           with p-value 3.80525e-33
Add  situ                           with p-value 4.07753e-28
Add  hrmn                           with p-value 9.03766e-22
Add  lum                            with p-value 2.74298e-23
Add  plan                           with p-value 1.22628e-13
Add  dep                            with p-value 8.88005e-10
Add  mois                           with p-value 1.58218e-08
Add  surf                           with p-value 1.2383e-08
Add  place                          with p-value 5.92251e-05
Add  nbv                            with p-value 0.00422366
Add  larrout                        with p-value 0.00871278
resulting features:
['obs', 'an', 'catv', 'catu', 'sexe', 'col', 'age', 'manv', 'obsm', 'agg', 'secu', 'situ', 'hrmn',
'lum', 'plan', 'dep', 'mois', 'surf', 'place', 'nbv', 'larrout']
```

In [ ]:

```
#On va garder seulement 'sexe', 'col', 'age', 'manv', 'obsm', 'agg', 'secu', 'hrmn', 'lum', 'plan'
, 'dep', 'mois', 'surf', 'place', 'senc', 'infra', 'situ'
```

In [207]:

```
X_train2,X_test2,y_train2,y_test2=train_test_split(x1[['sexe', 'col', 'age', 'manv', 'obsm', 'agg',
'secu', 'situ', 'hrmn', 'lum', 'plan', 'dep', 'mois', 'surf', 'place', 'nbv', 'larrout']],y1,test_s
ize=0.25,random_state=0)
```

In [208]:

```
logreg2 = LogisticRegression()
logreg2.fit(X_train2,y_train2)
```

Out[208]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [200]:

```
y_pred2=logreg2.predict(X_test2)
```

In [218]:

```
logreg2.score(x1[['sexe', 'col', 'age', 'manv', 'obsm', 'agg', 'secu', 'situ', 'hrmn', 'lum', 'plan
', 'dep', 'mois', 'surf', 'place', 'nbv', 'larrout']],y1)
```

Out[218]:

```
0.5012764832124658
```

In [202]:

```
print('Intercept: \n',logreg2.intercept_)
print('Coefficients: \n',logreg2.coef_)
```

```
Intercept:
 [-0.00107757 -0.00508964  0.00117307 -0.00550273]
Coefficients:
 [[-1.51207406e-01 -1.83187328e-02 -7.21567142e-02 -1.07757251e-03
  -4.19760528e-01  1.57260156e-01  1.13563947e-01  2.13487765e-02
   1.26978000e-01  4.94762699e-01  3.92326761e-01 -2.47347702e-01
   1.31632498e-04 -2.10895531e-01 -8.70469449e-02  3.33656623e-03
   1.10279472e-02 -5.48699123e-02 -3.17434027e-01  4.68657551e-02
   4.05923564e-05]
 [ 3.91866428e-02 -8.65238615e-02  2.18257953e-02 -5.08963891e-03
  -6.04671428e-01 -5.46523710e-02  2.32725478e-01  6.74971386e-03
  -1.55565687e-01 -1.55267654e+00 -7.80503423e-01  3.93547629e-01
  -1.95338527e-04  3.96369682e-01  1.29311263e-01 -3.99984823e-03
   4.17241714e-03  9.65913670e-03 -1.94882061e-02 -1.86259024e-01
   8.38508600e-04]
 [ 7.73112441e-02  1.99421436e-02  3.88881546e-02  1.17306727e-03
  -8.23167106e-02 -1.08343988e-01 -1.11951407e-02 -9.00656878e-03
  -1.25589147e-01 -7.84988593e-01 -3.16282402e-01  2.86963731e-01
  -6.97783428e-06  1.10165822e-01  4.44483075e-02 -7.34435686e-03
   5.37816228e-03  2.80718526e-02  7.67036451e-03 -2.10466815e-01
   4.12376021e-04]
 [ 3.63106662e-02 -9.35464018e-02  2.84168746e-02 -5.50272952e-03
   5.13397939e-01 -9.50488719e-02 -1.49565558e-01 -1.76003479e-02
  -4.87978033e-02  2.12231029e-01  6.95669102e-03 -8.80199103e-02
  -1.17319666e-04  1.08668008e-01  2.62519834e-02  2.38991607e-03
  -1.37119745e-02  2.86383694e-02  4.45573047e-03  1.13057534e-01
  -4.43130059e-04]]
```

In [209]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test2,y_pred2))
```

```
             precision    recall  f1-score   support

          1       0.60      0.90      0.72     11761
          2       0.00      0.00      0.00       625
          3       0.49      0.24      0.32      4624
          4       0.52      0.33      0.40      7570

avg / total       0.54      0.58      0.53     24580


C:\Users\Njeimi Amal\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [213]:

```
cnf_matrix2 = metrics.confusion_matrix(y_test2, y_pred2)
```

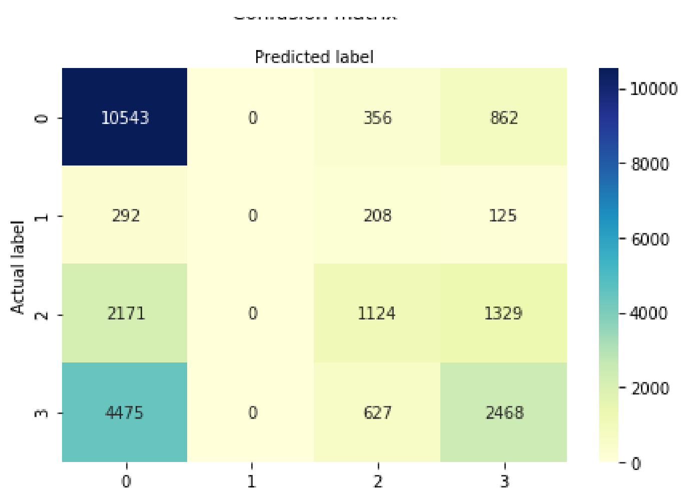
In [214]:

```
class_names2=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names2))
plt.xticks(tick_marks, class_names2)
plt.yticks(tick_marks, class_names2)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix2), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[214]:

```
Text(0.5,257.44,'Predicted label')
```

Confusion matrix



Segmentation

déterminer le nombre optimal de clusters pour k-means

In [24]:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

In [43]:

```
df3=pd.DataFrame(df2)
```

In [61]:

```
#on va supprimer l'adresse et la voie puisque on a déja la lat et long des lieux
df3.drop(["adr","voie"],axis=1,inplace=True)
```

In [51]:

```
df3.head()
```

Out[51]:

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp
0	201700000001	0.0	7	0	0.0	2.0	3.0	9.0	17	1	...	1.0	1	3	1	9.0	0.0	0.0	0.0
2	201700000001	0.0	10	0	0.0	2.0	3.0	13.0	17	1	...	1.0	1	3	1	1.0	0.0	0.0	0.0
3	201700000002	0.0	7	0	0.0	0.0	1.0	16.0	17	2	...	1.0	1	1	1	0.0	1.0	0.0	0.0
4	201700000002	0.0	1	0	0.0	0.0	7.0	1.0	17	2	...	1.0	1	3	1	5.0	0.0	0.0	0.0
5	201700000003	0.0	10	0	0.0	2.0	1.0	1.0	17	3	...	1.0	1	1	1	1.0	1.0	0.0	0.0

5 rows × 44 columns

In [55]:

```
df3["gps"]=df3["gps"].replace([1,2,3,4,5],["M","A","G","R","Y"], inplace=True)
```

In [56]:

```
df3["gps"] = df3.gps.astype(float)
```

In [37]:

```
round(df3,2)
df3.head()
```

Out[37]:

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	place	catu	grav	sexe	trajet	secu	locp	actp
0	2.017000e+11	0.0	7.0	0.0	0.0	2.0	3.0	9.0	17.0	1.0	...	1.0	1.0	3.0	1.0	9.0	0.0	0.0	0.0
2	2.017000e+11	0.0	10.0	0.0	0.0	2.0	3.0	13.0	17.0	1.0	...	1.0	1.0	3.0	1.0	1.0	0.0	0.0	0.0
3	2.017000e+11	0.0	7.0	0.0	0.0	0.0	1.0	16.0	17.0	2.0	...	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0
4	2.017000e+11	0.0	1.0	0.0	0.0	0.0	7.0	1.0	17.0	2.0	...	1.0	1.0	3.0	1.0	5.0	0.0	0.0	0.0
5	2.017000e+11	0.0	10.0	0.0	0.0	2.0	1.0	1.0	17.0	3.0	...	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0

5 rows × 44 columns

In [57]:

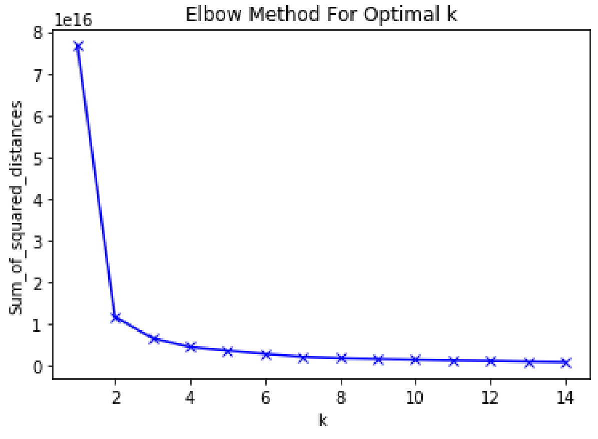
```
df3=df3.astype('float')
```

In [78]:

```
Sum_of_squared_distances = []
K = range(1,15)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(df3.iloc[:,1:df3.shape[1]])
    Sum_of_squared_distances.append(km.inertia_)
```

In [63]:

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



Dans le graphique ci-dessus, le coude est à k = 4, ce qui indique que k optimal 4.

In [92]:

```
#Cluster K-means
Mkmeans=KMeans(n_clusters=4)
#adapter le modèle de données
Mkmeans.fit(df3.iloc[:,1:df3.shape[1]])
```

In [82]:

```
df3['cluster']=Mkmeans.fit_predict(df3.iloc[:,1:df3.shape[1]])
```

```
df3['cluster']=Mkmeans.fit_predict(df3.iloc[:,1:df3.shape[1]])
```

In [83]:

```
df3.sort_values(by="cluster").tail()
```

Out[83]:

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	an	mois	...	catu	grav	sexe	trajet	secu	locp	actp	e
55384	2.017000e+11	0.0	7.0	0.0	0.0	2.0	1.0	16.0	17.0	3.0	...	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0
55383	2.017000e+11	0.0	7.0	0.0	0.0	2.0	6.0	1.0	17.0	3.0	...	1.0	1.0	2.0	9.0	1.0	0.0	0.0	0
55382	2.017000e+11	1.0	2.0	0.0	0.0	2.0	2.0	16.0	17.0	3.0	...	1.0	4.0	1.0	9.0	1.0	0.0	0.0	0
55479	2.017000e+11	0.0	10.0	0.0	0.0	1.0	3.0	1.0	17.0	3.0	...	1.0	1.0	2.0	1.0	1.0	0.0	0.0	0
69328	2.017000e+11	2.0	7.0	0.0	0.0	2.0	8.0	9.0	17.0	12.0	...	1.0	1.0	2.0	5.0	1.0	0.0	0.0	0

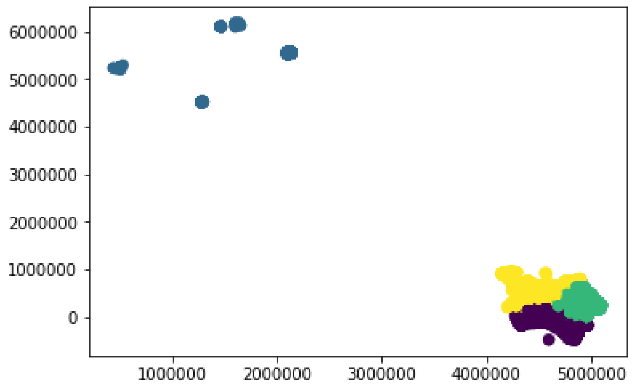
5 rows × 44 columns

In [104]:

```
colormap=np.array(['Red','green','blue','red'])
plt.scatter(df3.lat, df3.long, c=Mkmeans.labels_, s=40)
```

Out[104]:

```
<matplotlib.collections.PathCollection at 0x2c983a39588>
```



In [ ]:

```
c1=df3[df3.cluster==0]
c2=df3[df3.cluster==1]
c3=df3[df3.cluster==2]
c4=df3[df3.cluster==3]
```

In [ ]:

```
#c1.sort_values(by="lat",ascending=True)
#c1.sort_values(by="long",ascending=True)
#lat 4555195.0 - 5107423.0
#long -54782.0 -  699102.0
```

In [ ]:

```
#c2.sort_values(by="lat",ascending=True)
#c2.sort_values(by="long",ascending=True)
#lat 434091.0 - 2138567.0
#long 4505711.0 - 6179073.0
```

In [ ]:

```
#c3.sort_values(by="lat",ascending=True)
```

```
#c3.sort_values(by="long",ascending=True)
#lat 4147009.0 - 4915682.0
#long 179166.0 - 954584.0
```

In [ ]:

```
#c4.sort_values(by="lat",ascending=True)
#c4.sort_values(by="long",ascending=True)
#lat 4275118.0 - 24970738.0
#long -477098.0 - 231383.0
```

Donc cette segmentation nous a permis de distinguer les zones géographiques suivant

Cluster 1 </br>

lattitude entre 4555195 et 5107423 </br>

longitude entre -54782 et 699102

Cluster 2 </br>

lattitude entre 434091 et 2138567 </br>

longitude entre 4505711 et 6179073

Cluster 3 </br>

lattitude entre 4147009 et 4915682 </br>

longitude entre 179166 et 954584

Cluster 4 </br>

lattitude entre 4275118 et 24970738 </br>

longitude entre -477098 et 231383