In [1]:
```
# Importation des librairies
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from joblib import dump
```

In [28]:
```
# Importation des jeu de données
events = pd.read_csv('events.csv')
parentid = pd.read_csv('category_tree.csv')
properties1= pd.read_csv('item_properties_part1.csv')
properties2= pd.read_csv('item_properties_part2.csv')
# On rassemble les deux dataframe - properties1 et properties2
properties = pd.concat([properties1, properties2], ignore_index=Tru
```

In [29]:
```
### EXPLORATION DES DONNÉES
```

In [31]:
```
# VISUALISATION DU DATAFRAME 'events'
```

In [32]:
```
events
```

Out[32]:

	timestamp	visitorid	event	itemid	transactionid
0	1433221332117	257597	view	355908	NaN
1	1433224214164	992329	view	248676	NaN
2	1433221999827	111016	view	318965	NaN
3	1433221955914	483717	view	253185	NaN
4	1433221337106	951259	view	367447	NaN
...	...	...	...	...	...
2756096	1438398785939	591435	view	261427	NaN
2756097	1438399813142	762376	view	115946	NaN
2756098	1438397820527	1251746	view	78144	NaN
2756099	1438398530703	1184451	view	283392	NaN
2756100	1438400163914	199536	view	152913	NaN

2756101 rows × 5 columns

In [33]:
```
"""
le 'visitorid' est l'utilisateur unique qui a navigué sur le site w
L'evenement 'event' est ce que l'utilisateur a fait durant sa visit
Les produits 'itemid' correpondent à l'ensemble des produits qui on
Le 'transactionid' n'aura de valeur que si l'utilisateur a effectué
La variables timestamp correspond à la date à laquelle chaque inter
"""
```

Out[33]:
```
"\nle 'visitorid' est l'utilisateur unique qui a navigué sur le si
te web. \nL'evenement 'event' est ce que l'utilisateur a fait dura
nt sa visite sur le web : vue / ajout au panier / transation\nLes
produits 'itemid' correpondent à l'ensemble des produits qui ont n
écessité une interaction avec l'utilisateur. \nLe 'transactionid'
n'aura de valeur que si l'utilisateur a effectué un achat, c'est p
our cela qu'on observe à l'oeil nu beaucoup de NaN. \nLa variables
timestamp correspond à la date à laquelle chaque interaction a été
effectuée sur la pariode observée.\n"
```

In [34]:
```
events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2756101 entries, 0 to 2756100
Data columns (total 5 columns):
 #   Column         Dtype  
---  ------         -----  
 0   timestamp      int64  
 1   visitorid      int64  
 2   event          object 
 3   itemid         int64  
 4   transactionid  float64
dtypes: float64(1), int64(3), object(1)
memory usage: 105.1+ MB
```

In [35]:
```
# Visiteurs total vs visiteurs uniques
print('Visiteurs total:', len(events['visitorid']))
print('Visiteurs uniques:', len(events['visitorid'].unique()))
```

```
Visiteurs total: 2756101
Visiteurs uniques: 1407580
```

In [36]:
```
# Comportement des visiteurs sur le site web
# Une visualisation précise est effectué sur le visiteur qui a effe
#events['visitorid'].value_counts()
events.loc[events.visitorid == 1150086]
```

Out[36]:

	timestamp	visitorid	event	itemid	transactionid
226726	1434074901572	1150086	view	49925	NaN
226815	1434077292739	1150086	transaction	332592	7821.0
226997	1434046293048	1150086	view	408132	NaN
227022	1434044566676	1150086	view	109352	NaN
227094	1434047479464	1150086	view	107195	NaN
...	...	...	...	...	...
2755097	1438385978884	1150086	view	163848	NaN
2755127	1438379057318	1150086	view	190000	NaN
2755295	1438378300182	1150086	view	441852	NaN
2755369	1438379122123	1150086	view	110529	NaN
2755410	1438357399949	1150086	view	6913	NaN

7757 rows × 5 columns

In [37]:
```
"""
On peut comprendre que certains utilisateurs ont visités le site qu
Ainsi, le visiteur '1150086' a effectué 7 757 interactions.
"""
```

Out[37]:
```
"\nOn peut comprendre que certains utilisateurs ont visités le sit
e qu'une seule fois. Tandis que d'autres, ont effectué un grand no
mbre d'interaction (ci-dessus). \nAinsi, le visiteur '1150086' a e
ffectué 7 757 interactions.\n"
```

In [38]:
```
# Voici un autre exemple pour illustrer cela
# comportement de l'acheteur ID227091
events[(events.visitorid == 227091) & (events.itemid == 346212)].so
```

Out[38]:

	timestamp	visitorid	event	itemid	transactionid
1845234	1432233114809	227091	view	346212	NaN
1855845	1432233450583	227091	addtocart	346212	NaN
1846471	1432233512989	227091	transaction	346212	9042.0
1870076	1432319622352	227091	view	346212	NaN
1876439	1432322306344	227091	view	346212	NaN

In [39]:
```
"""
Ce visiteur a effectué le chemin classique de l'évènement en regard
le produit. Par ailleurs, on constate également que celui-ci est re
le produit qu'il a déjà acheté.
"""
```

Out[39]:
```
"\nCe visiteur a effectué le chemin classique de l'évènement en re
gardant le produit, l'ajoutant à son panier, puis achète \nle prod
uit. Par ailleurs, on constate également que celui-ci est revenu d
eux fois sur le site pour de nouveau regarder \nle produit qu'il a
déjà acheté. \n"
```

In [40]:
```
# Détaille sur les transactions effectuées sur le site web
print("Nombre d'achateurs uniques: ", len(events[events.transaction
print("Nombre total des achats en volume: ", len(events[events.tran
```

```
Nombre d'achateurs uniques:  11719
Nombre total des achats en volume:  22457
```

In [41]:
```
"""
Très peu d'achats sont effectués par rapport au nombre de vues obse
"""
```

Out[41]:
```
" \nTrès peu d'achats sont effectués par rapport au nombre de vues
observées sur toute la période. \n"
```

In [42]:
```
# Détaille sur les produits du site web
print("Nombre de produits total: ",events['itemid'].nunique())
print("Nombre de produits vendus: ",len(events[events.transactionid
```

```
Nombre de produits total:  235061
Nombre de produits vendus:  12025
```

In [43]:
```
"""
On constate que l'entreprise dispose d'un portefeuille produit très
Différentes interprétations sont possibles, à savoir si l'ensemble
sont biens visiblent auprès des utilisateurs. En outre, est-ce que
des utilisateurs.
"""
```

Out[43]:
```
"\nOn constate que l'entreprise dispose d'un portefeuille produit
très élevé. Cependant, un grand nombre d'entre eux n'ont pas été a
chetés par les clients.\nDifférentes interprétations sont possible
s, à savoir si l'ensemble des produits disponibles sur le site e-c
ommerce \nsont biens visiblent auprès des utilisateurs. En outre,
est-ce que tous les produits sont pertinents et répondent aux beso
ins \ndes utilisateurs. \n"
```

In [44]:
```
# Nombre de valeurs manquantes au sein de la variable 'transactioni
print("Nombre de valeurs manquantes de la variable 'transactionid'
"""
Ce montant correspond à la totalité des NaN présents dans la variab
"""
```

```
Nombre de valeurs manquantes de la variable 'transactionid' : 2733
644
```

Out[44]:
```
"\nCe montant correspond à la totalité des NaN présents dans la va
riable 'transactionid'.\n"
```

In [45]:
```
# Détaille de la variable de l'évènement 'event'
events['event'].value_counts()
```

Out[45]:
```
view           2664312
addtocart        69332
transaction      22457
Name: event, dtype: int64
```

In [46]:
```
"""
La répartition des données de la variable 'event' est totalement dé
également pour les modèles prédictifs qui seront développés plus ta
"""
```

Out[46]:
```
"\nLa répartition des données de la variable 'event' est totalemen
t déséquilibrée. De ce fait, il faudra le prendre en compte dans l
a visualisation des données, mais\négalement pour les modèles préd
ictifs qui seront développés plus tard dans la syntaxe.\n"
```

In [47]:
```
# VISUALISATION DU DATAFRAME 'parentid'
```

In [48]:
```
parentid
```

Out[48]:

	categoryid	parentid
0	1016	213.0
1	809	169.0
2	570	9.0
3	1691	885.0
4	536	1691.0
...	...	...
1664	49	1125.0
1665	1112	630.0
1666	1336	745.0
1667	689	207.0
1668	761	395.0

1669 rows × 2 columns

In [49]:
```
# Détaille des categoryid et des parentid
print("Nombre de categoryid:", parentid['categoryid'].nunique())
print("Nombre de parentid:", parentid['parentid'].nunique())
"""
Au sein de ce dataframe, nous observons l'enfant categorid et le pa
"""
```

```
Nombre de categoryid: 1669
Nombre de parentid: 362
```

Out[49]:
```
"\nAu sein de ce dataframe, nous observons l'enfant categorid et l
e parent correspondant. On observe que pour 1 669 category, il y a
uniquement 362 parentid. \n"
```

In [50]:
```
# VISUALISATION DU DATAFRAME PROPERTIES
properties
```

Out[50]:

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
1	1441508400000	206783	888	1116713 960601 n277.200
2	1439089200000	395014	400	n552.000 639502 n720.000 424566
3	1431226800000	59481	790	n15360.000
4	1431831600000	156781	917	828513
...	...	...	...	...
20275897	1433646000000	236931	929	n12.000
20275898	1440903600000	455746	6	150169 639134
20275899	1439694000000	347565	686	610834
20275900	1433646000000	287231	867	769062
20275901	1442113200000	275768	888	888666 n10800.000 746840 1318567

20275902 rows × 4 columns

In [51]:
```
"""
la propriété d'un article peut varier dans le temps (par exemple, l
Toutes les valeurs du fichier à l'exception des propriétés « catego
La valeur de la propriété "categoryid" contient l'identifiant de ca
La valeur de la propriété "disponible" contient la disponibilité de
Toutes les valeurs numériques ont été marquées d'un caractère "n" a
"""
```

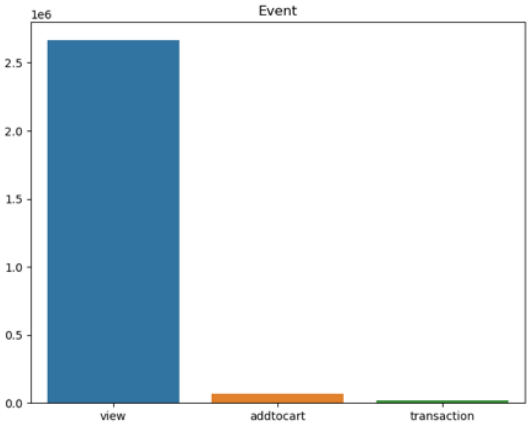
Out[51]:
```
'\nla propriété d\'un article peut varier dans le temps (par exemp
le, les changements de prix dans le temps), chaque ligne du fichie
r a un horodatage correspondant.\nToutes les valeurs du fichier à
l\'exception des propriétés « categoryid » et « available », ont é
té hachées.\nLa valeur de la propriété "categoryid" contient l\'id
entifiant de catégorie d\'élément.\nLa valeur de la propriété "dis
ponible" contient la disponibilité de l\'article, c\'est-à-dire qu
e 1 signifie que l\'article était disponible, sinon 0. \nToutes le
s valeurs numériques ont été marquées d\'un caractère "n" au début
et ont une précision de 3 chiffres après la virgule décimale, par
exemple, " 5" deviendra "n5.000",\n'
```

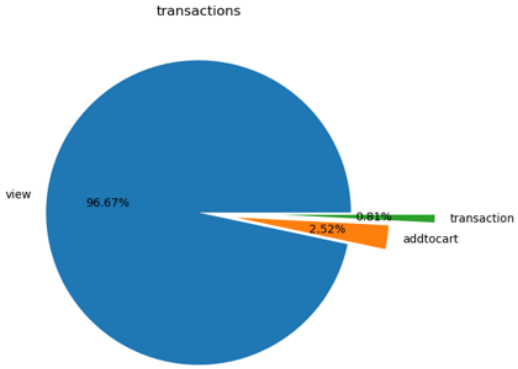
In [ ]:
```
### VISUALISATION DES DONNÉES
```

In [52]:
```
#Le but est d'avoir un aperçu général de l'evenement 'event' qui es
#'View' / 'Addtocary' / 'Transaction'

events_count = events["event"].value_counts()
events_count
```

Out[52]:
```
view           2664312
addtocart        69332
transaction      22457
Name: event, dtype: int64
```

In [53]:
```
plt.figure(figsize=(17,6))
plt.subplot(1,2,1)
sns.barplot(x=events_count.index, y=events_count.values)
plt.title('Event')

plt.subplot(1,2,2)
plt.title('transactions')
explode = (0, 0.25, 0.55)
plt.pie(events_count.values, explode=explode, labels=events_count.i
plt.show()
```





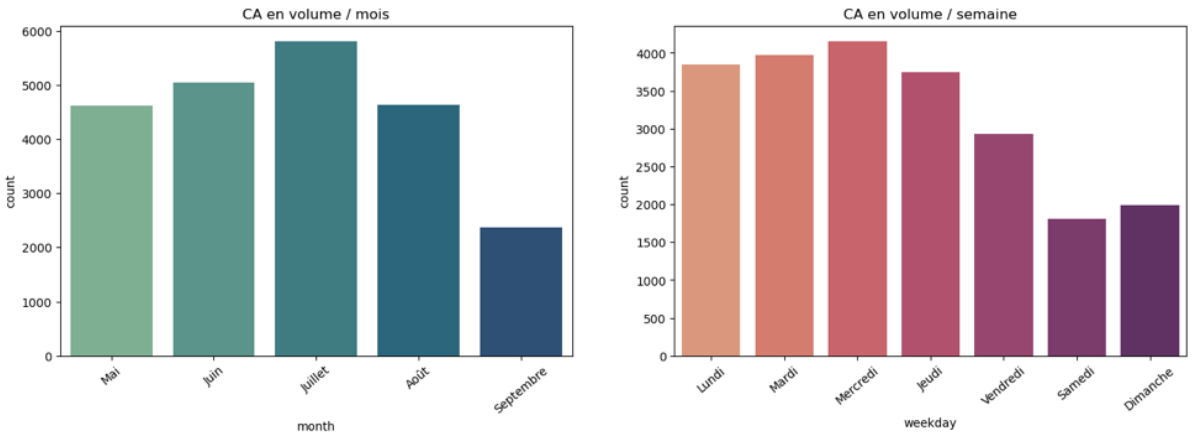
In [54]:
```
# Il est possible d'analyser les transactions en fonction du temps.
# L'analyse s'effectuera à différentes échelles : mois, semaine et

# La variable 'timestamp' se doit d'être convertie en datetime, afi
# à travers le temps

import datetime
events['timestamp_bis']=pd.to_datetime(events['timestamp'],unit='ms
events['date'] = events['timestamp_bis'].dt.date
events['hour'] = events['timestamp_bis'].dt.hour
events['month'] = events['timestamp_bis'].dt.month
events['weekday'] = events['timestamp_bis'].dt.weekday
events.head()
#events.info()
```

Out[54]:

	timestamp	visitorid	event	itemid	transactionid	timestamp_bis	date	hour	mon
0	1433221332117	257597	view	355908	NaN	2015-06-02 05:02:12.117	2015-06-02	5	
1	1433224214164	992329	view	248676	NaN	2015-06-02 05:50:14.164	2015-06-02	5	
2	1433221999827	111016	view	318965	NaN	2015-06-02 05:13:19.827	2015-06-02	5	
3	1433221955914	483717	view	253185	NaN	2015-06-02 05:12:35.914	2015-06-02	5	
4	1433221337106	951259	view	367447	NaN	2015-06-02 05:02:17.106	2015-06-02	5	

In [56]:
```
plt.figure(figsize=(17,5))

plt.subplot(1,2,1)
sns.countplot(x='month', data = events[events['event'] == 'transact
plt.xticks(np.arange(5), ['Mai', 'Juin', 'Juillet', 'Août', 'Septem
plt.xticks(rotation=40)
plt.title("CA en volume / mois")

plt.subplot(1,2,2)
sns.countplot(x='weekday', data = events[events['event'] == 'transa
plt.xticks(np.arange(7), ['Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'V
plt.xticks(rotation=40)
plt.title("CA en volume / semaine");

print("La période d'activité des données se situent entre le 3 Mai
```

```
La période d'activité des données se situent entre le 3 Mai 2015 e
t le 18 Sept 2015
```



In [57]:
```
"""
Puisque nous avons uniquement à disposition la moitié des données d
des informations de ce mois pour obtenir un mois entier.
"""
```

Out[57]:
```
"\nPuisque nous avons uniquement à disposition la moitié des donné
es du mois de Septembre, nous allons copier l'ensemble \ndes infor
mations de ce mois pour obtenir un mois entier.\n"
```

In [58]:
```
# Dupliquer les valeurs du mois de Septembre pour obtenir un mois c
# On va tout d'abord sélectionner uniquement le mois de Septembre '
sept = events[events['month'] == 9]
sept
```

Out[58]:

	timestamp	visitorid	event	itemid	transactionid	timestamp_bis	date	hou
1145993	1441083472374	138734	view	139298	NaN	2015-09-01 04:57:52.374	2015-09-01	4
1145994	1441083380262	938236	view	374698	NaN	2015-09-01 04:56:20.262	2015-09-01	4
1145995	1441086029058	1016904	view	294319	NaN	2015-09-01 05:40:29.058	2015-09-01	5
1145996	1441083571340	349857	view	277328	NaN	2015-09-01 04:59:31.340	2015-09-01	4
1145997	1441085078078	843815	view	42267	NaN	2015-09-01 05:24:38.078	2015-09-01	5
...	...	...	...	...	...	...	...	..
1462514	1442541481522	1088606	view	82126	NaN	2015-09-18 01:58:01.522	2015-09-18	1
1462515	1442541962785	444612	view	256931	NaN	2015-09-18 02:06:02.785	2015-09-18	2
1462516	1442542019902	1128231	view	202067	NaN	2015-09-18 02:06:59.902	2015-09-18	2
1462517	1442540868781	157029	view	454888	NaN	2015-09-18 01:47:48.781	2015-09-18	1
1462518	1442540856491	1177658	view	281710	NaN	2015-09-18 01:47:36.491	2015-09-18	1

303710 rows × 10 columns

In [59]:
```
# Un nouveau dataframe est créé, contenant les données copiées
sept_duplicated = sept.copy()
# On effectue une concaténation du df events avec la variable conte
events_sept = pd.concat([events, sept_duplicated], ignore_index=Tru
events_sept
```

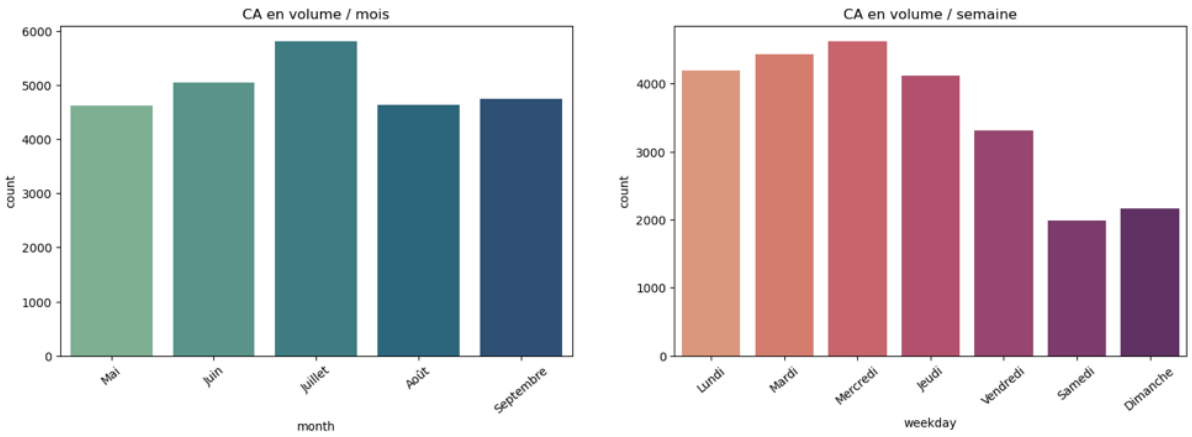
Out[59]:

	timestamp	visitorid	event	itemid	transactionid	timestamp_bis	date	hou
0	1433221332117	257597	view	355908	NaN	2015-06-02 05:02:12.117	2015-06-02	5
1	1433224214164	992329	view	248676	NaN	2015-06-02 05:50:14.164	2015-06-02	5
2	1433221999827	111016	view	318965	NaN	2015-06-02 05:13:19.827	2015-06-02	5
3	1433221955914	483717	view	253185	NaN	2015-06-02 05:12:35.914	2015-06-02	5
4	1433221337106	951259	view	367447	NaN	2015-06-02 05:02:17.106	2015-06-02	5
...	...	...	...	...	...	...	...	..
3059806	1442541481522	1088606	view	82126	NaN	2015-09-18 01:58:01.522	2015-09-18	1
3059807	1442541962785	444612	view	256931	NaN	2015-09-18 02:06:02.785	2015-09-18	2
3059808	1442542019902	1128231	view	202067	NaN	2015-09-18 02:06:59.902	2015-09-18	2
3059809	1442540868781	157029	view	454888	NaN	2015-09-18 01:47:48.781	2015-09-18	1
3059810	1442540856491	1177658	view	281710	NaN	2015-09-18 01:47:36.491	2015-09-18	1

3059811 rows × 10 columns

In [60]:
```
# On visualise de nouveau les résultats
plt.figure(figsize=(17,5))

plt.subplot(1,2,1)
sns.countplot(x='month', data = events_sept[events_sept['event'] ==
plt.xticks(np.arange(5), ['Mai', 'Juin', 'Juillet', 'Août', 'Septem
plt.xticks(rotation=40)
plt.title("CA en volume / mois")

plt.subplot(1,2,2)
sns.countplot(x='weekday', data = events_sept[events_sept['event'] =
plt.xticks(np.arange(7), ['Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'V
plt.xticks(rotation=40)
plt.title("CA en volume / semaine");

print("La période d'activité des données se situent entre le 1 Mai
```

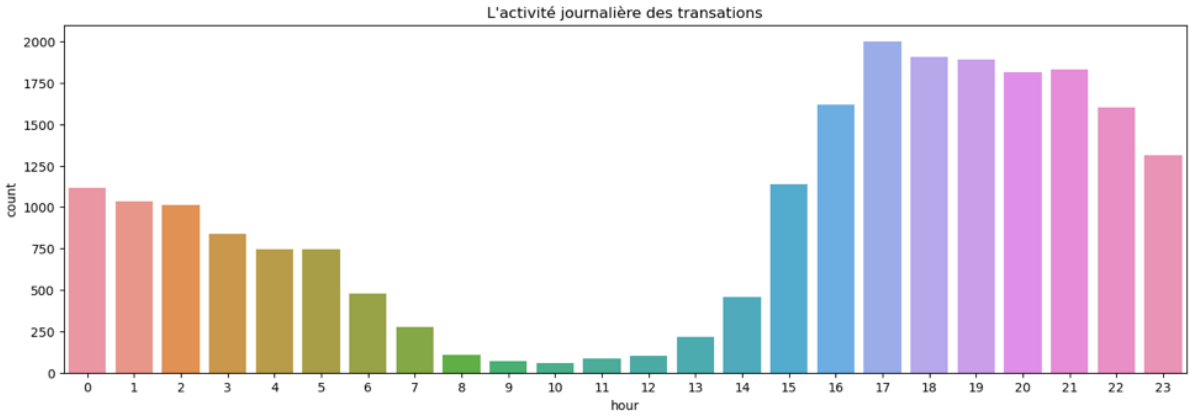
```
La période d'activité des données se situent entre le 1 Mai 2015 e
t le 18 Sept 2015
```



In [64]:
```
"""
Après traitement, on observe que le mois de Septembre constitue le
On peut supposer que la période de Juin et Juillet constitue les so
naturellement.

Concernant l'activité hebdomadaire, il est intéressant de constater
en début de semaine et non le week-end. Malheureusement, nous n'avo
de l'entreprise pour comprendre ce phénomène.
"""
```

Out[64]:
```
"\nAprès traitement, on observe que le mois de Septembre constitue
le 3ème meilleur mois, après Juillet et Juin. \nOn peut supposer q
ue la période de Juin et Juillet constitue les soldes. De ce fait,
les ventes en volume augmentent \nnaturellement. \n\nConcernant l'
activité hebdomadaire, il est intéressant de constater que les tra
nsactions, s'effectuent majoritairement \nen début de semaine et n
on le week-end. Malheureusement, nous n'avons pas d'informations c
omplémentaires sur les produits \nde l'entreprise pour comprendre
ce phénomène. \n"
```

In [62]:
```
# Visualisation de l'activité journalière

plt.figure(figsize=(16,5))

sns.countplot(x='hour', data = events[events['event'] == 'transacti
plt.title("L'activité journalière des transations");
```



In [63]:
```
"""
On peut supposer que la période creuse est la période au cours de l
une faible activité. En revanche, l'activité la plus forte commence
Sachant que nous analysons un site e-commerce, cela permet à l'entr
l'activité est soutenue. En outre, on observe également une moyenne
du matin. Par conséquent, c'est une période à ne pas négliger et qu
"""
```

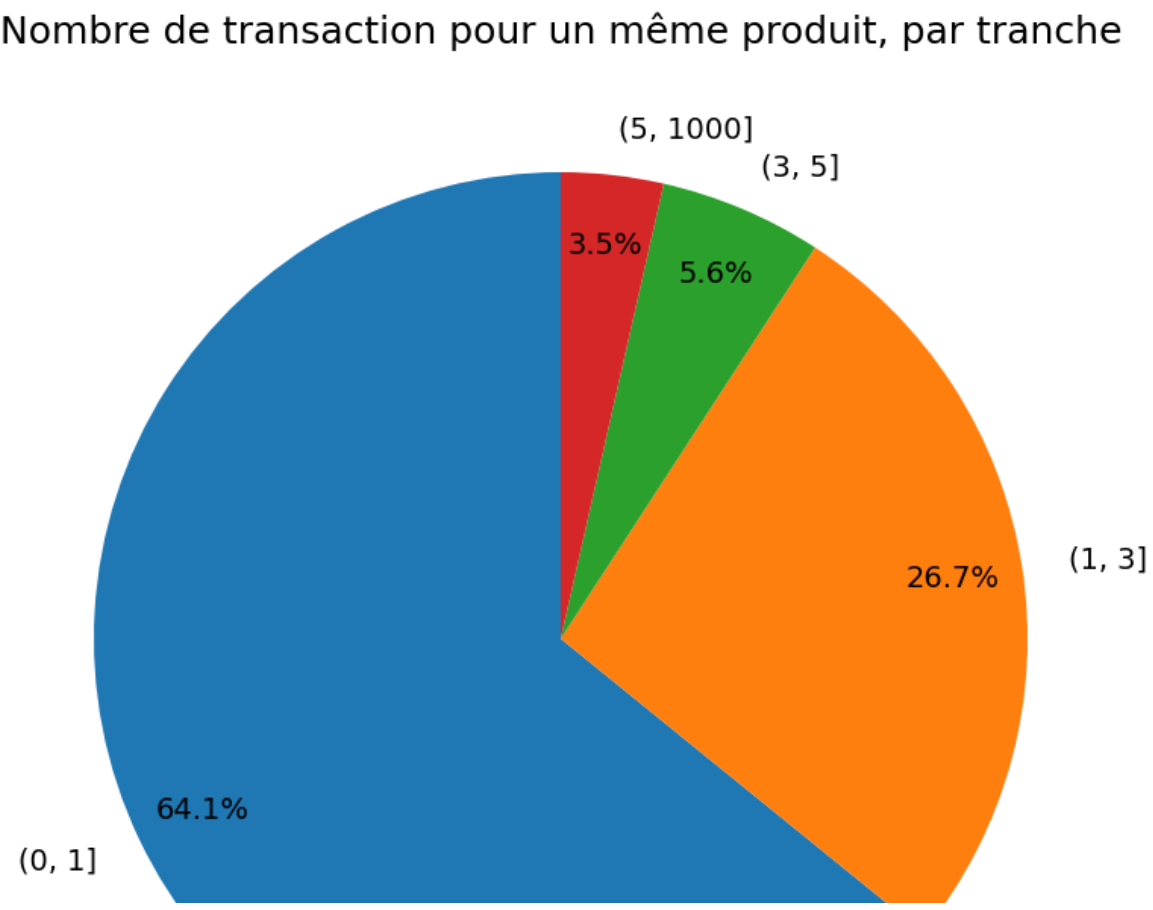
Out[63]:
```
"\nOn peut supposer que la période creuse est la période au cours
de laquelle les gens travaillent. C'est pourquoi, on observe \nune
faible activité. En revanche, l'activité la plus forte commence en
début d'après-midi, s'étalant jusqu'au soir. \nSachant que nous an
alysons un site e-commerce, cela permet à l'entreprise d'accentuer
sa communication aux heures ou \nl'activité est soutenue. En outre
, on observe également une moyenne de 800 transactions par heure,
entre minuits et cinq heures\ndu matin. Par conséquent, c'est une
période à ne pas négliger et que l'entreprise doit absolument expl
oiter.\n"
```

In [65]:
```
# Step 1 : A partir du DF "events", je place dans un nouveau DF "Tr
# En procédant ainsi je ne retiens que les évènements "transactions

transac    = (events[(events["event"]=="transaction")])

# Step 2 : Je créé un troisième DF "top_vente" dédié à ne retenir q
# attention, le nombre de ces itemid doit être de 22 457. Ce nombre

top_vente  = pd.DataFrame(columns = {"itemid_vente", "itemid_vente_
top_vente["itemid_vente"]         = pd.Series(transac["itemid"])

# Step 3 : Dans un quatrième DF "Nbre_vente" je compte le nombre d'
# générait des NaN faussant les résultats.

Nbre_vente = top_vente["itemid_vente"].value_counts()

# Step 4 : créé un index afin de récupérer la colonne avec les item

Nbre_vente = Nbre_vente.reset_index()

# Step 5 : renommer les colonnes

Nbre_vente = Nbre_vente.rename(columns = ({"itemid_vente" : "itemid

# Step 6 : Appliquer un filtre sur le nombre de transaction pour la

Nbre_vente_sorted = Nbre_vente[(Nbre_vente["itemid_vente_count"]>20

#Step 7 : Vérification de l'intégrité des données
print("Le nombre de transaction doit être de :",Nbre_vente["itemid_
Nbre_vente["itemid_vente_count"].sort_values(ascending=False)
print("La totalité des transaction a été réalisées sur",Nbre_vente[
```

```
Le nombre de transaction doit être de : 22457
La totalité des transaction a été réalisées sur 38 item_id différe
nts
```

In [66]:

```
Nbre_vente["Tranche"] = pd.cut(Nbre_vente["itemid_vente_count"],
           bins= [0,1,3,5,1000],
           right=True,
           labels=None,
           retbins=False,
           precision=3,
           include_lowest=False,
           duplicates='raise',
           ordered=True)

plt.figure(figsize = (15,10))
plt.pie(Nbre_vente['Tranche'].value_counts(),
        labels=Nbre_vente['Tranche'].value_counts().index,
        startangle=90, autopct='%.1f%%',
        pctdistance=0.85,
        textprops={'fontsize': 14})

plt.title("Nombre de transaction pour un même produit, par tranche"
plt.show()
"""
Ce graphique démontre que très peu de produits sont vendus plusieur
transactions récurrentes.
En d'autres termes, le site ne peut pas s'appuyer sur des produits

On constate par ce graphique que l'on doit appliquer un seuil pour
Remarques: peut être est ce du à une volonté du site qui fait tourn
"""
```



Out[66]: '\nCe graphique démontre que très peu de produits sont vendus plus ieurs fois. Il n\'y a donc pas de produit "phare" qui constituerai t des \ntransactions récurrentes.\nEn d\'autres termes, le site ne peut pas s\'appuyer sur des produits "phares" pour générer un chif fre d\'affaires récurrent et stable.\n\nOn constate par ce graphiq ue que l\'on doit appliquer un seuil pour une correcte visualisati on.\nRemarques: peut être est ce du à une volonté du site qui fait tourner ses produits ? -> Analyser le categoryid Available pour dé terminer ce point.\n'

In [67]:
```
#idem pour l'évènement view

view = events[(events["event"]=="view")]
top_view                      = pd.DataFrame(columns = {"itemid_vie
top_view["itemid_view"]       = pd.Series(view["itemid"])
Nbre_view = top_view["itemid_view"].value_counts()


Nbre_view = Nbre_view.reset_index()

Nbre_view = Nbre_view.rename(columns = ({"itemid_view" : "itemid_vi

Nbre_view_sorted = Nbre_view[(Nbre_view["itemid_view_count"]>1000)]

print("Le nombre de transaction doit être de :",Nbre_view["itemid_v
Nbre_view["itemid_view_count"].sort_values(ascending=False)
print("La totalité des transaction a été réalisées sur",Nbre_view["
```

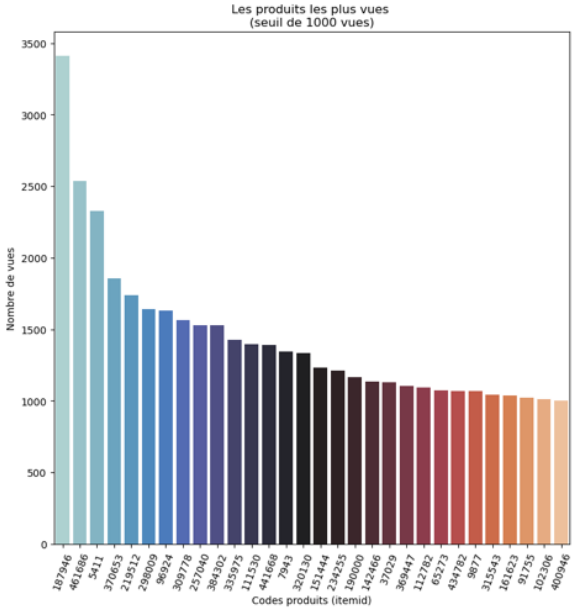
```
Le nombre de transaction doit être de : 2664312
La totalité des transaction a été réalisées sur 577 item_id différ
ents
```

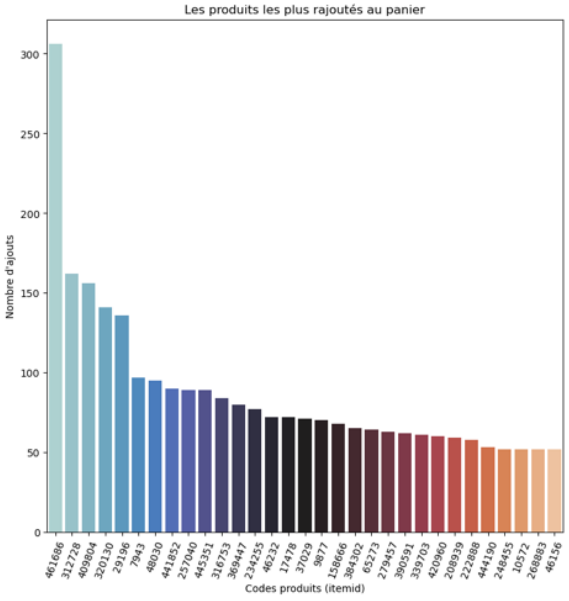
In [68]:
```
# Analyse sur les ajouts

add_to_cart                 = (events[(events["event"]=="addtocart"
top_add                     = pd.DataFrame(columns = {"itemid_add",
top_add["itemid_add"]         = pd.Series(add_to_cart["itemid"])

Nbre_add = top_add["itemid_add"].value_counts()
Nbre_add = Nbre_add.reset_index()
Nbre_add = Nbre_add.rename(columns = ({"itemid_add" : "itemid_add_c

Nbre_add_sorted = Nbre_add[(Nbre_add["itemid_add_count"]>50)]
print("Le nombre d'ajout doit être de :",Nbre_add["itemid_add_count
Nbre_add["itemid_add_count"].sort_values(ascending=False)
print("La totalité des ajouts a été réalisée sur",Nbre_add["itemid_
```

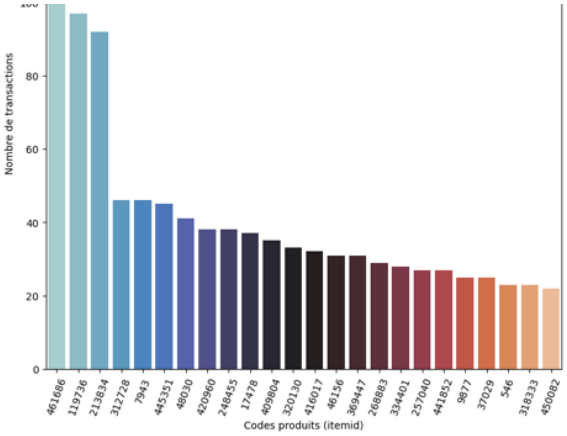
```
Le nombre d'ajout doit être de : 69332
La totalité des ajouts a été réalisée sur 74 item_id différents
```

In [69]:

```
plt.figure(figsize=(20,20))

plt.subplot(2,2,1)
sns.barplot(x = Nbre_view_sorted["itemid_view"],
            y = Nbre_view_sorted["itemid_view_count"],
            order =  Nbre_view_sorted["itemid_view"],
            palette = "icefire");
plt.xticks(rotation = 70)
plt.title("Les produits les plus vues \n(seuil de 1000 vues)")
plt.ylabel("Nombre de vues")
plt.xlabel("Codes produits (itemid)");

plt.subplot(2,2,2)
sns.barplot(x= Nbre_add_sorted["itemid_add"],
            y= Nbre_add_sorted["itemid_add_count"],
            order = Nbre_add_sorted["itemid_add"],
            palette = "icefire");
plt.xticks(rotation = 70)
plt.title("Les produits les plus rajoutés au panier")
plt.ylabel("Nombre d'ajouts")
plt.xlabel("Codes produits (itemid)");

plt.subplot(2,2,3)
sns.barplot(x= Nbre_vente_sorted["itemid_vente"],
            y =  Nbre_vente_sorted["itemid_vente_count"],
            order = Nbre_vente_sorted["itemid_vente"],
            palette = "icefire");
plt.xticks(rotation = 70)
plt.title("Les produits les plus vendus \n(seuil de 20 transactions
plt.ylabel("Nombre de transactions")
plt.xlabel("Codes produits (itemid)");
```





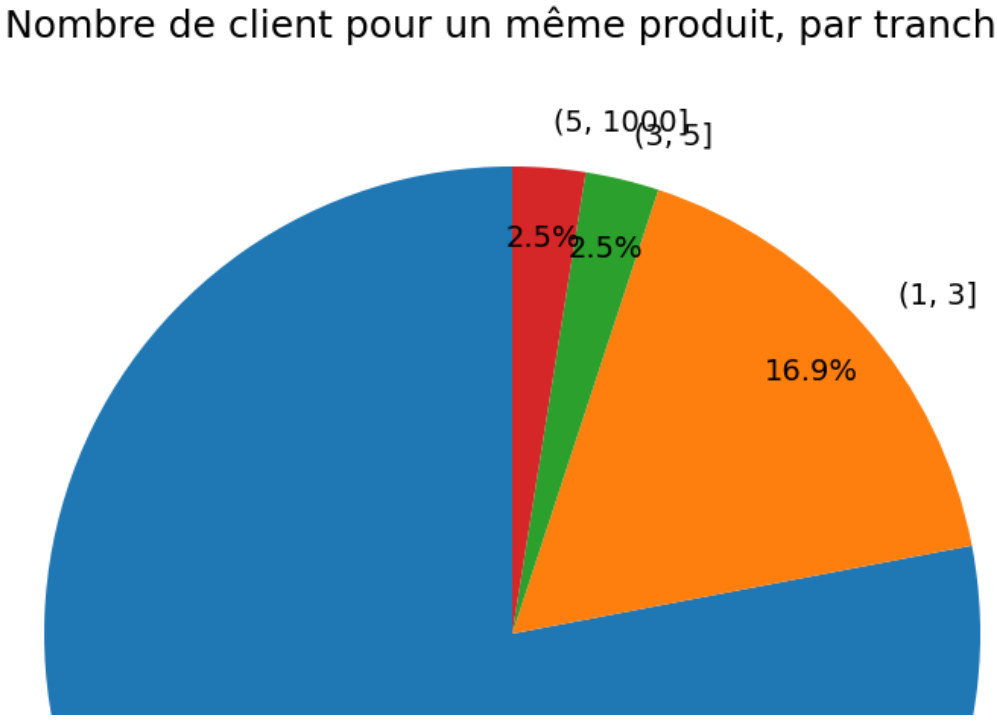


In [70]:
```
# Step 1 : A partir du DF "events", je places dans un nouveau DF "T
# En procédant ainsi je ne retiens que les évènements "transactions

transac    = (events[(events["event"]=="transaction")])

# Step 2 : Je créé un troisièdme DF "top_vente" dédié à ne retenir
# attention, le nombre de ces itemid doit être de 22 457. Ce nombre

top_vente_client  = pd.DataFrame(columns = {"visitorid_vente", "vis
top_vente_client["visitorid_vente"]         = pd.Series(transac["vi

# Step 3 : Dans un quatrième DF "Nbre_vente" je compte le nombre d'
# générait des NaN faussant les résultats.

Nbre_vente_client = top_vente_client["visitorid_vente"].value_count

# Step 4 : créé un index afin de récupérer la colonne avec les item

Nbre_vente_client = Nbre_vente_client.reset_index()

# Step 5 : renommer les colonnes

Nbre_vente_client = Nbre_vente_client.rename(columns = ({"visitorid

# Step 6 : Appliquer un filtre sur le nombre de transaction pour la

Nbre_vente_sorted_client = Nbre_vente_client[(Nbre_vente_client["vi

#Step 7 : Vérification de l'intégrité des données
print("Le nombre de transaction doit être de :",Nbre_vente_client["
Nbre_vente_client["visitorid_vente_count"].sort_values(ascending=Fa
print("La totalité des transaction a été réalisées sur",Nbre_vente_
```

```
Le nombre de transaction doit être de : 22457
La totalité des transaction a été réalisées sur 76 visitor_id diff
érents
```

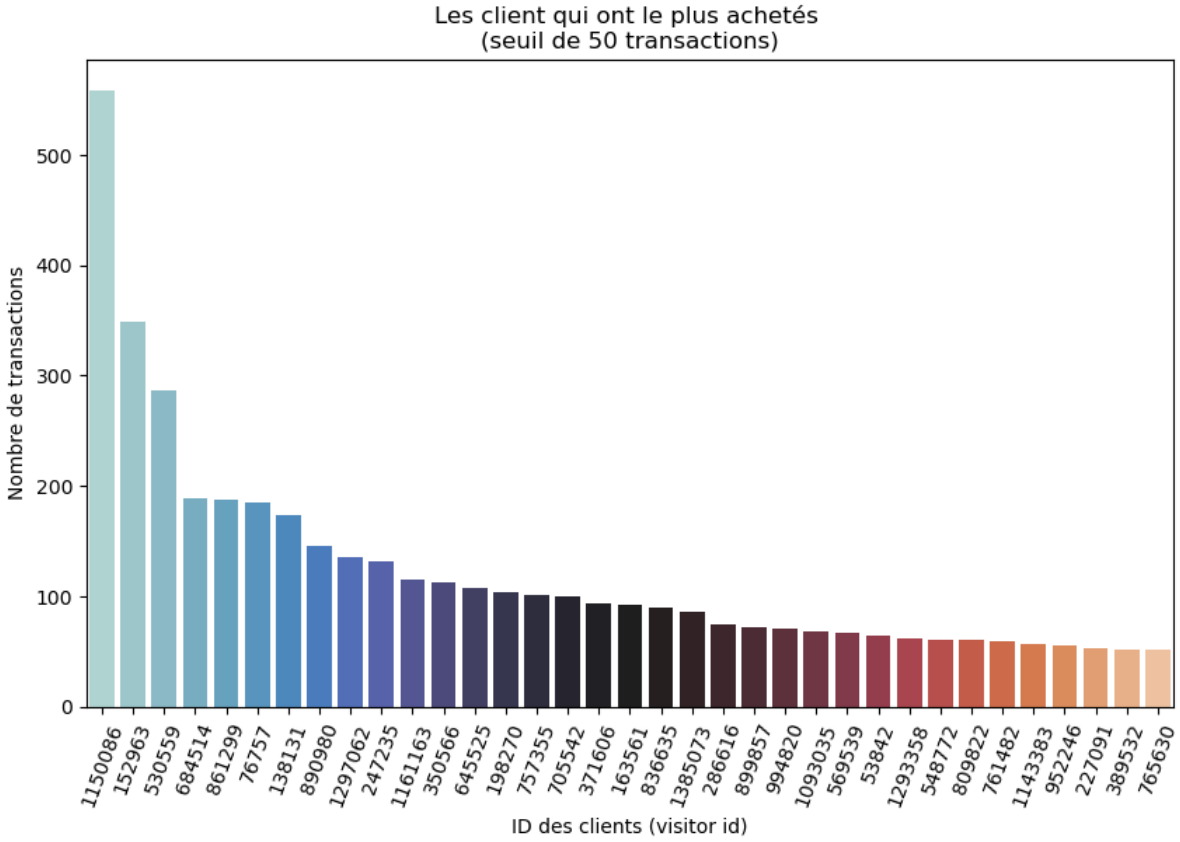
In [71]:
```
```

```
Nbre_vente_client["Tranche"] = pd.cut(Nbre_vente_client["visitorid_
            bins= [0,1,3,5,1000],
           right=True,
           labels=None,
           retbins=False,
           precision=3,
           include_lowest=False,
           duplicates='raise',
           ordered=True)

plt.figure(figsize = (15,10))
plt.pie(Nbre_vente_client["Tranche"].value_counts(),
        labels=Nbre_vente_client["Tranche"].value_counts().index,
        startangle=90, autopct='%.1f%%',
        pctdistance=0.85,
        textprops={'fontsize': 14})

plt.title("Nombre de client pour un même produit, par tranche", fon
plt.show()

#sns.countplot(x="Tranche", data = Nbre_vente_client, palette = "ic
#plt.xticks(rotation = 70);
#plt.title("Répartition du nombre de vente par client, par tranche)
#plt.ylabel("Nombre de vente pour un même client");
#plt.xlabel("Tranches");

"""
On constate par ce graphique que l'on doit appliquer un seuil pour
Par ailleurs cela démontre que la plupart des produits ne sont vend
On en déduit que le site internet ne vend pas des produits de grand
issu de la grande distribution (Leclerc.fr, monoprix.fr etc...)

"""
```



Out[71]: "\nOn constate par ce graphique que l'on doit appliquer un seuil p
our une correcte visualisation.\nPar ailleurs cela démontre que la
plupart des produits ne sont vendus qu'une seule fois à un même cl
ient.\nOn en déduit que le site internet ne vend pas des produits
de grande consommation. Il ne s'agit donc pas d'un site\nissu de l
a grande distribution (Leclerc.fr, monoprix.fr etc...)\n\n"

In [76]:
```
plt.figure(figsize=(10,6))
sns.barplot(x= Nbre_vente_sorted_client["visitorid_vente"],
            y= Nbre_vente_sorted_client["visitorid_vente_count"],
            order= Nbre_vente_sorted_client["visitorid_vente"],
            palette = "icefire");
plt.xticks(rotation = 70)
plt.title("Les client qui ont le plus achetés \n(seuil de 50 transa
plt.ylabel("Nombre de transactions")
plt.xlabel("ID des clients (visitor id)");
```



In [78]:
```
"""
Nous constatons que la grande majorité des clients n'ont pas réalis
"""
```

Out[78]:
```
"\nNous constatons que la grande majorité des clients n'ont pas ré
alisé plus de 50 transactions sur l'intervalle temporel étudié.\n"
```

In [ ]:
```
### Création du dataset qui sera utilisé pour les modèles prédictif
```

In [2]:
```
# Importation des jeux de données
events = pd.read_csv('events.csv')
parentid = pd.read_csv('category_tree.csv')
properties1= pd.read_csv('item_properties_part1.csv')
properties2= pd.read_csv('item_properties_part2.csv')
# On rassemble les deux dataframe - properties1 et properties2
properties = pd.concat([properties1, properties2], ignore_index=Tru
```

In [ ]:
```
# Avant d'aller plus loin, revisualisons les dataframe pour compren
```

In [149]:
```
# df 'events'
events
```

Out[149]:

	timestamp	visitorid	event	itemid	transactionid
0	1433221332117	257597	view	355908	NaN
1	1433224214164	992329	view	248676	NaN
2	1433221999827	111016	view	318965	NaN
3	1433221955914	483717	view	253185	NaN
4	1433221337106	951259	view	367447	NaN
...	...	...	...	...	...
2756096	1438398785939	591435	view	261427	NaN
2756097	1438399813142	762376	view	115946	NaN
2756098	1438397820527	1251746	view	78144	NaN
2756099	1438398530703	1184451	view	283392	NaN
2756100	1438400163914	199536	view	152913	NaN

2756101 rows × 5 columns

In [150]:
```
# df 'properties'
properties
```

Out[150]:

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
1	1441508400000	206783	888	1116713 960601 n277.200
2	1439089200000	395014	400	n552.000 639502 n720.000 424566
3	1431226800000	59481	790	n15360.000
4	1431831600000	156781	917	828513
...	...	...	...	...
20275897	1433646000000	236931	929	n12.000
20275898	1440903600000	455746	6	150169 639134
20275899	1439694000000	347565	686	610834
20275900	1433646000000	287231	867	769062
20275901	1442113200000	275768	888	888666 n10800.000 746840 1318567

20275902 rows × 4 columns

In [151]:
```
# df 'parentid'
parentid
```

Out[151]:

	categoryid	parentid
0	1016	213.0
1	809	169.0
2	570	9.0
3	1691	885.0
4	536	1691.0
...	...	...
1664	49	1125.0
1665	1112	630.0
1666	1336	745.0
1667	689	207.0
1668	761	395.0

1669 rows × 2 columns

In [152]:
```
# Comprehension des variables 'property' et value' du df properties

# À travers les informations disponiblent sur kaggle, on nous infor
# ont été hachées et que seul les modalités 'categoryid' et 'availa

# Concernant la variable 'value', toutes les valeurs numériques ont
# au début et ont une précision de 3 chiffres après la virgule déci
```

In [153]:
```
# Les valeurs 790 et 888 contiennent le plus d'occurence dans la co
display(properties['property'].value_counts())
```

```
888           3000398
790           1790516
available     1503639
categoryid     788214
6              631471
               ...
782                 1
288                 1
722                 1
744                 1
769                 1
Name: property, Length: 1104, dtype: int64
```

In [154]:
```
# Visualisation de la valeur 790
properties_790 = properties[properties['property'] == '790']
properties_790.head(20)
#properties_790.tail(20)
```

Out[154]:

	timestamp	itemid	property	value
3	1431226800000	59481	790	n15360.000
14	1434250800000	169055	790	n21000.000
16	1435460400000	178601	790	n5400.000
42	1431831600000	125874	790	n39588.000
46	1433646000000	272201	790	n10320.000
55	1432436400000	407811	790	n185280.000
63	1433646000000	119637	790	n11160.000
70	1437274800000	459523	790	n15588.000
99	1434250800000	181493	790	n191736.000
103	1433646000000	439391	790	n54048.000
114	1431831600000	103584	790	n459360.000
120	1439694000000	16499	790	n252000.000
123	1434250800000	56337	790	n23160.000
125	1436670000000	8840	790	n94800.000
142	1433646000000	97792	790	n65280.000
144	1437274800000	95163	790	n38400.000
145	1436670000000	28212	790	n400080.000
146	1431831600000	459171	790	n108120.000
147	1437879600000	86554	790	n11520.000
157	1431831600000	127727	790	n296496.000

In [155]:
```
# Visualisation de la valeur 888
properties_888 = properties[properties['property'] == '888']
properties_888.head(20)
#properties_790.tail(20)
```

Out[155]:

	timestamp	itemid	property	value
1	1441508400000	206783	888	1116713 960601 n277.200
10	1439089200000	450113	888	1038400 45956 n504.000
17	1436670000000	319291	888	1292080
26	1435460400000	16615	888	150169 176547 824301 24474 293011 1240134
40	1433646000000	152892	888	599031
45	1435460400000	95237	888	883447 726612
48	1442113200000	372355	888	1051803 1227205 n712862568.000 992862 951748 6...
57	1434250800000	4454	888	1024724 220869
65	1432436400000	313836	888	n120.000 586893 1166722 237874 43137
75	1431226800000	53090	888	297660 1301543
77	1440903600000	74996	888	113942 492465 n168.000
93	1436065200000	387064	888	543080 702002 n48.000
98	1436065200000	26690	888	824508 n7536.000 309080 594060 892470
104	1431831600000	329408	888	1333963 747375 927741 1071593 547687 199524
109	1437274800000	41111	888	150169 460346 174342 750333 523033 33148
113	1434250800000	211525	888	236689 1037087 1318567
126	1431831600000	131258	888	665993
139	1440298800000	197394	888	722389 543387 679871 n180.000 1175087 n144.000...
143	1439089200000	53057	888	140639 419119
148	1439089200000	213101	888	1207060 620840

In [156]:
```
"""
Après visualisation, on constate que toutes les valeurs numérique m
de la valeur 790. On peut en déduire que la modalité 790 représente
"""
```

Out[156]:
```
'\nAprès visualisation, on constate que toutes les valeurs numériq
ue marquées d\'un caractère "n" au début, proviennent\nde la valeu
r 790. On peut en déduire que la modalité 790 représente le prix i
nitial des produits. \n'
```

In [157]:
```
# Traitement du dataset
```

In [3]:
```
# On sélectionne uniquement les éléments pertinents que l'on veut g

# Le premier critère est basé sur la colonne "property" du DataFram
# des valeurs "categoryid", "790" ou "available". Cela est fait en
# un masque booléen pour chaque ligne où la condition est vraie.

# La deuxième étape permet de conserver uniquement les lignes du df
# colonne 'itemid' est présente dans la colonne 'itemid' du df 'eve

properties = properties[properties.property.isin(["categoryid","790"
                        properties.itemid.isin(events.itemid.unique
properties
```

Out[3]:

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
3	1431226800000	59481	790	n15360.000
5	1436065200000	285026	available	0
14	1434250800000	169055	790	n21000.000
15	1437274800000	186518	available	0
...	...	...	...	...
20275872	1435460400000	444741	categoryid	511
20275876	1436670000000	147935	790	n42720.000
20275889	1433041200000	356167	available	0
20275890	1432436400000	206640	790	n9600.000
20275891	1439089200000	200211	available	0

2669516 rows × 4 columns

In [4]:
```
# Nettoyage de la variable 'value' - On supprime les caractères 'n'
# pour obtenir les prix. Par la suite, on transforme les données en

properties.value = properties.value.str.replace("n","").astype("flo
properties
```

Out[4]:

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338.0
3	1431226800000	59481	790	15360.0
5	1436065200000	285026	available	0.0
14	1434250800000	169055	790	21000.0
15	1437274800000	186518	available	0.0
...	...	...	...	...
20275872	1435460400000	444741	categoryid	511.0
20275876	1436670000000	147935	790	42720.0
20275889	1433041200000	356167	available	0.0
20275890	1432436400000	206640	790	9600.0
20275891	1439089200000	200211	available	0.0

2669516 rows × 4 columns

In [5]:
```
# On renomme les 3 modalités pour obtenir un df propre et compréhen
properties.property = properties.property.map({"790":"price","categ
                                               "available":"availab
properties
```

Out[5]:

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338.0
3	1431226800000	59481	price	15360.0
5	1436065200000	285026	available	0.0
14	1434250800000	169055	price	21000.0
15	1437274800000	186518	available	0.0
...	...	...	...	...
20275872	1435460400000	444741	categoryid	511.0
20275876	1436670000000	147935	price	42720.0
20275889	1433041200000	356167	available	0.0
20275890	1432436400000	206640	price	9600.0
20275891	1439089200000	200211	available	0.0

2669516 rows × 4 columns

In [ ]:
```
# Nettoyage du df events
```

In [6]:
```
# Sélection du DataFrame events en ne conservant que les lignes pou
# de la colonne itemid est présente dans la colonne itemid du DataF
events = events[events.itemid.isin(properties.itemid.unique())]
events
```

Out[6]:

	timestamp	visitorid	event	itemid	transactionid
0	1433221332117	257597	view	355908	NaN
1	1433224214164	992329	view	248676	NaN
3	1433221955914	483717	view	253185	NaN
4	1433221337106	951259	view	367447	NaN
5	1433224086234	972639	view	22556	NaN
...	...	...	...	...	...
2756096	1438398785939	591435	view	261427	NaN
2756097	1438399813142	762376	view	115946	NaN
2756098	1438397820527	1251746	view	78144	NaN
2756099	1438398530703	1184451	view	283392	NaN
2756100	1438400163914	199536	view	152913	NaN

2500516 rows × 5 columns

In [86]:
```
### Former la validité du df properties
```

In [7]:
```
# Trouver le timestamp maximal parmi les deux df events et properti
# timestamp du df properties dans l'ordre croissant.
max_timestamp = np.max((events.timestamp.max(),properties.timestamp
properties_sorted = properties.sort_values("timestamp")

# Valeur de la propriété précédente:
# On ajoute une colonne "lag_value" qui contient les valeurs de la
# d'une position vers la gauche pour chaque groupe formé par les co
properties_sorted["lag_value"] = properties_sorted.groupby(["itemid'

# Temps du prochain changement (temps de fin de validité):
# On ajoute une colonne "lead_timestamp" qui contient les valeurs d
# décalées d'une position vers la droite pour chaque groupe formé p
properties_sorted["lead_timestamp"] = properties_sorted.groupby(["i

# Observation des changements possibles:
# On ajoute une colonne "is_change" qui contient des valeurs boolée
# colonne "value" est différente de la valeur de la colonne "lag_va
# colonne "lag_value" est manquante.
properties_sorted["is_change"] = np.logical_or(properties_sorted.la
    properties_sorted.lag_value!=properties_sorted.value)
# On utilise une notation de filtrage pour sélectionner uniquement
# Cela signifie que seuls les changements de valeurs seront conserv
properties_sorted = properties_sorted[properties_sorted.is_change]
```

```
properties_sorted = properties_sorted[properties_sorted.is_change]

# Temps du prochain changement (temps de fin de validité)
properties_sorted["lead_timestamp"] = properties_sorted.groupby(["i

# Remplir les valeurs manquantes:
# La première fonction utilise la méthode fillna pour remplacer les
# la colonne "lead_timestamp" par une valeur spécifique (max_timest
# On utilise l'option "inplace=True" pour que les modifications soi
# sur le dataframe et non sur une copie.
properties_sorted["lead_timestamp"].fillna(max_timestamp, inplace=T
# On convertie la colonne en nombre entier
properties_sorted["lead_timestamp"] = properties_sorted["lead_times
    .astype("int64")

# Renommer certaines variables:
properties_sorted.rename({"timestamp":"valid_start",
    "lead_timestamp":"valid_end"}, axis=1, inplace=True)

# Effectuer un filtre sur les colonnes:
properties = properties_sorted.loc[:,("valid_start", "valid_end",
    "itemid", "property", "value")]

# Ajouter un temps valide:
properties["time_valid"] = properties.valid_end - properties.valid_

# Suppression des variables dont on n'a plus besoin:
del properties_sorted, max_timestamp
properties.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1022834 entries, 14619574 to 6570229
Data columns (total 6 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   valid_start  1022834 non-null  int64
 1   valid_end    1022834 non-null  int64
 2   itemid       1022834 non-null  int64
 3   property     1022834 non-null  object
 4   value        1022834 non-null  float64
 5   time_valid   1022834 non-null  int64
dtypes: float64(1), int64(4), object(1)
memory usage: 54.6+ MB
```

In [8]:
```
# Aperçu du traitement effectué sur la variable 'properties'
properties
```

Out[8]:

	valid_start	valid_end	itemid	property	value	time_valid
14619574	1431226800000	1431831600000	202983	available	0.0	604800000
2075505	1431226800000	1442545187788	317740	available	0.0	11318387788
7660150	1431226800000	1432436400000	216549	available	1.0	1209600000
2075491	1431226800000	1442545187788	136500	price	63600.0	11318387788
17300300	1431226800000	1442545187788	337475	price	46584.0	11318387788
...	...	...	...	...	...	...
4261807	1442113200000	1442545187788	128188	price	493200.0	431987788
11964186	1442113200000	1442545187788	303993	price	13680.0	431987788
5684490	1442113200000	1442545187788	418638	categoryid	208.0	431987788
11964452	1442113200000	1442545187788	460681	price	7080.0	431987788
6570229	1442113200000	1442545187788	61888	price	26880.0	431987788

1022834 rows × 6 columns

In [9]:
```
# Rajouter le df 'properties' à la df 'events' à l'aide de la varia

events_enhanced = events.merge(properties, on="itemid")

# On utilise une notation de filtrage pour sélectionner uniquement
# où la valeur de la colonne "timestamp" est supérieure ou égale à
# de la colonne "valid_start" et inférieure à la valeur de la colon
# On utilise la fonction numpy.logical_and pour combiner les deux c
# les colonnes "timestamp", "valid_start" et "valid_end". Au final,
# une valeur de temps qui est dans l'intervalle défini par les colo
# dans le dataframe "events_enhanced".
events_enhanced = events_enhanced[
np.logical_and(events_enhanced.timestamp>=events_enhanced.valid_sta
        events_enhanced.timestamp<events_enhanced.valid_end)]

# Sélection des variables
events_enhanced = events_enhanced.loc[:,["timestamp", "visitorid",
    "event", "property", "value"]]

events_enhanced = events_enhanced.pivot_table(
    index=["timestamp","visitorid","itemid", "event"],
    columns="property", values="value",
    observed=True)

events_enhanced.columns = list(events_enhanced.columns)
events_enhanced = events_enhanced.reset_index()
events_enhanced.rename(index={"property":"index"},
    inplace=True)

# On supprime la variable dont on n'a plus besoin
del events
# Observation de valeurs manquantes
events_enhanced.isnull().sum()
```

Out[9]:
```
timestamp          0
visitorid          0
itemid             0
event              0
available     127804
categoryid    252920
price          83443
dtype: int64
```

In [165]:
```
"""
Nous constatons un nombre important de valeurs manquantes dans les
Pour résoudre ce problème, nous avons décidé d'imputer les NaNs ave
étendue.
"""
```

Out[165]:
```
"\nNous constatons un nombre important de valeurs manquantes dans
les colonnes 'categoryid', 'price' et 'available'. \nPour résoudre
ce problème, nous avons décidé d'imputer les NaNs avec des valeurs
valides sur la période la plus\nétendue.\n"
```

In [166]:
```
# Remplir les valeurs manquantes
```

In [10]:
```
# Obtenir les propriétés valides le plus longtemps
top_properties = properties.groupby(["itemid","property","value"],
    as_index=False).time_valid.sum().sort_values("time_valid")\
        .groupby(["itemid","property"]).tail(1)

top_properties = top_properties.pivot_table(index=["itemid"],
    columns="property", values="value", observed=True).reset_index(

# Remplir les valeurs manqantes lorsque c'est nécessaire
events_enhanced = events_enhanced.merge(top_properties, on="itemid"

events_enhanced.loc[events_enhanced.categoryid_x.isna(),
    ["categoryid_x"]] = events_enhanced["categoryid_y"]

events_enhanced.loc[events_enhanced.price_x.isna(),
    ["price_x"]] = events_enhanced["price_y"]

events_enhanced.loc[events_enhanced.available_x.isna(),
    ["available_x"]] = events_enhanced["available_y"]

# Renommer les variables en question
events_enhanced.rename({"categoryid_x":"categoryid",
                        "price_x":"price",
                        "available_x":"available"},
                        axis=1, inplace=True)

# Positionnement des variables dans le nouveau df 'event_enhanced'
events_enhanced = events_enhanced.loc[:,["timestamp","visitorid","i
    "event", "categoryid","available", "price"]]

# Suppression des variables dont on n'a plus besoin
del top_properties, properties
```

In [11]:
```
# Vérifier s'il y a toujours des valeurs manquantes
events_enhanced.isnull().sum()
```

Out[11]:
```
timestamp     0
visitorid     0
itemid        0
event         0
categoryid    0
available     0
price         0
dtype: int64
```

In [12]:
```
# Visulisation du nouveau df
events_enhanced
```

Out[12]:

	timestamp	visitorid	itemid	event	categoryid	available	price
0	1431226809663	350663	325406	view	1051.0	0.0	189600.0
1	1432859215323	837591	325406	view	1051.0	0.0	189600.0
2	1440602227285	1280398	325406	view	1051.0	0.0	189600.0
3	1440953791096	31391	325406	view	1051.0	0.0	189600.0
4	1431226810854	826796	40046	view	921.0	1.0	12540.0
...	...	...	...	...	...	...	...
2351688	1442544490530	250797	48374	view	1483.0	0.0	1498800.0
2351689	1442544902961	571995	448372	view	1366.0	0.0	7680.0
2351690	1442545034410	369667	306478	view	421.0	0.0	3480.0
2351691	1442545137779	485144	14990	view	646.0	0.0	186960.0
2351692	1442545164029	472345	301436	view	1244.0	0.0	21120.0

2351693 rows × 7 columns

In [13]:
```
# Traitement de la variable 'timestamp' afin d'avoir différentes in
import datetime

events_enhanced['timestamp']=pd.to_datetime(events_enhanced['timest
events_enhanced['date'] = events_enhanced['timestamp'].dt.date
events_enhanced['hour'] = events_enhanced['timestamp'].dt.hour
events_enhanced['month'] = events_enhanced['timestamp'].dt.month
events_enhanced['week'] = events_enhanced['timestamp'].dt.week
events_enhanced['weekday'] = events_enhanced['timestamp'].dt.weekda

events_enhanced.head()
#events.info()
```

```
/var/folders/s3/78mw79ln4456kxpb8mlqhs8h0000gn/T/ipykernel_11022/1
516626149.py:8: FutureWarning: Series.dt.weekofyear and Series.dt.
week have been deprecated. Please use Series.dt.isocalendar().week
instead.
  events_enhanced['week'] = events_enhanced['timestamp'].dt.week
```

Out[13]:

	timestamp	visitorid	itemid	event	categoryid	available	price	date	hour	mont
0	2015-05-10 03:00:09.663	350663	325406	view	1051.0	0.0	189600.0	2015-05-10	3	
1	2015-05-29 00:26:55.323	837591	325406	view	1051.0	0.0	189600.0	2015-05-29	0	
2	2015-08-26 15:17:07.285	1280398	325406	view	1051.0	0.0	189600.0	2015-08-26	15	
3	2015-08-30 16:56:31.096	31391	325406	view	1051.0	0.0	189600.0	2015-08-30	16	
4	2015-05-10 03:00:10.854	826796	40046	view	921.0	1.0	12540.0	2015-05-10	3	

In [14]:
```
# Rajout de la variable parentid du df 'parentid'
events_enhanced= events_enhanced.merge(parentid, how="left", on=["c
display(events_enhanced.isna().sum())
```

```
timestamp      0
visitorid      0
itemid         0
event          0
categoryid     0
available      0
price          0
date           0
hour           0
month          0
week           0
weekday        0
parentid      15
dtype: int64
```

In [15]:
```
# Remplire les NaN restants de la colonne 'parentid' avec son mode
events_enhanced['parentid'].fillna(events_enhanced['parentid'].mode
display(events_enhanced.isna().sum())
```

```
timestamp     0
visitorid     0
itemid        0
event         0
categoryid    0
available     0
price         0
date          0
hour          0
month         0
week          0
weekday       0
parentid      0
dtype: int64
```

In [16]:
```
# Repositionnement des colonnes
events_enhanced = events_enhanced.reindex(columns=['timestamp','dat
                                                   'itemid','event'

events_enhanced.sort_values(['itemid','timestamp'], inplace=True)

events_enhanced
```

Out[16]:

	timestamp	date	month	week	weekday	hour	visitorid	itemid	event	availa
2317326	2015-08-18 18:30:40.493	2015-08-18	8	34	1	18	370720	3	view	
2317327	2015-08-31 14:39:02.792	2015-08-31	8	36	0	14	639016	3	view	
2163135	2015-06-30 07:03:11.545	2015-06-30	6	27	1	7	1042455	4	view	
2163136	2015-08-31 18:06:00.244	2015-08-31	8	36	0	18	905555	4	view	
2163137	2015-09-15 23:22:44.099	2015-09-15	9	38	1	23	1010132	4	view	
...	...	...	...	...	...	...	...	...	...	
510826	2015-08-11 07:22:11.087	2015-08-11	8	33	1	7	1388296	466864	view	
510827	2015-08-11 07:53:28.134	2015-08-11	8	33	1	7	1388296	466864	view	
510828	2015-08-12 00:28:32.082	2015-08-12	8	33	2	0	1388296	466864	view	
510829	2015-09-04 16:58:29.739	2015-09-04	9	36	4	16	770644	466864	view	
510830	2015-09-14 21:27:38.016	2015-09-14	9	38	0	21	948010	466864	view	

2351693 rows × 13 columns

In [24]:
```
### Modèles prédictifs
events_enhanced.to_csv('events_enhanced.csv', index=False)
```

In [175]:
```
"""
Notre variable cible est la variable "event". Nous sommes donc dans
cas d'apprentissage supervisé.

Cette variable possédant 3 modalités "view", "add", "transaction" i
s'agit donc d'une variable discrète.
Nous sommes sur une problématique de classification.
"""
```

Out[175]:
```
'\nNotre variable cible est la variable "event". Nous sommes donc
dans\ncas d\'apprentissage supervisé.\n\nCette variable possédant
3 modalités "view", "add", "transaction" il \ns\'agit donc d\'une
variable discrète.\nNous sommes sur une problématique de classific
ation.\n\n'
```

In [23]:
```
"""
Nous faisons face à un problème de classification déséquilibrée. En
la modalité 'view', le reste se répartit entre la modalité 'addtoca
L'utilisation de la moyenne géométrique (G-mean) s'avère utile pour
il s'agit de la racine du produit de la sensibilité et de la spécif
concentrerons davantage sur la fonction classification_report_imbal
de plusieurs métriques à disposition.

En somme, il existe deux méthodes principales que l'on peut utilise
Le sur-échantillonnage : Oversampling et le sous-échantillonnage :U

Les méthodes d'Oversampling fonctionnent en augmentant le nombre d'
classe(s) minoritaire(s) afin d'arriver à un ratio classe minoritai

Les méthodes d'Undersampling fonctionnent en diminuant le nombre d'
classe(s) majoritaire(s) afin d'arriver à un ratio classe minoritai
"""
```

Out[23]:
```
"\nNous faisons face à un problème de classification déséquilibrée
. En effet, environ 97% des données se situent dans \nla modalité
'view', le reste se répartit entre la modalité 'addtocart' et 'tra
nsaction'. \nL'utilisation de la moyenne géométrique (G-mean) s'av
ère utile pour les problèmes de classification déséquilibrée:\nil
s'agit de la racine du produit de la sensibilité et de la spécific
ité. C'est la raison pour laquelle nous nous \nconcentrerons davan
tage sur la fonction classification_report_imbalanced(), afin d'an
alyser les résultats à l'aide \nde plusieurs métriques à dispositi
on. \n\nEn somme, il existe deux méthodes principales que l'on peu
t utiliser pour égaliser les classes : \nLe sur-échantillonnage :
Oversampling et le sous-échantillonnage :Undersampling. \n\nLes mé
thodes d'Oversampling fonctionnent en augmentant le nombre d'obser
vations de la (des) \nclasse(s) minoritaire(s) afin d'arriver à un
ratio classe minoritaire/ classe majoritaire satisfaisant.\n\nLes
méthodes d'Undersampling fonctionnent en diminuant le nombre d'obs
ervations de la  (des) \nclasse(s) majoritaire(s) afin d'arriver à
un ratio classe minoritaire/ classe majoritaire satisfaisant. \n"
```

In [ ]:

In [17]:
```
# Suppression des colonnes qui ne sont pas jugées utiles
events_enhanced.drop(["timestamp", "date"], axis=1, inplace=True)
# verification si les colonnes sont bien supprimées
#events_enhanced
```

In [18]:
```
# Nommer dans un dataFrame X, les variables explicatives et y la va

X = events_enhanced.drop("event", axis = 1)

y = events_enhanced["event"]
```

In [19]:
```
# Division du dataset en jeu d'entrainement et en jeu de test

from sklearn.model_selection import train_test_split

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

In [20]:
```
# Transformation de la variable 'event'

# L'Encodage
# Les données à dispostion ne sont pas toujours des données numériq
# tous ces types de données nécessitent un traitement particulier p
# les algorithmes de Machine Learning prennent en entrée uniquement

# Comme explicité precedemment, la variable cible 'events' est de t
# celle-ci doit être converti en données numériques. Pour ce faire,
# traiter la variable.

# LabelEncoder:
from sklearn.preprocessing import LabelEncoder

#Instanciation du modèle
le = LabelEncoder()

y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
```

In [21]:
```
# Transformation des données

from sklearn.preprocessing import StandardScaler

#Instanciation du modèle
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [22]:
```
# Installation plus importation de la librairie imblearn
get_ipython().system('pip install imblearn')
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
```

```
Requirement already satisfied: imblearn in /Users/YannLadouceur/op
t/anaconda3/lib/python3.9/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /Users/YannLado
uceur/opt/anaconda3/lib/python3.9/site-packages (from imblearn) (0
.10.1)
Requirement already satisfied: numpy>=1.17.3 in /Users/YannLadouce
ur/opt/anaconda3/lib/python3.9/site-packages (from imbalanced-lear
n->imblearn) (1.21.5)
Requirement already satisfied: scikit-learn>=1.0.2 in /Users/YannL
adouceur/opt/anaconda3/lib/python3.9/site-packages (from imbalance
d-learn->imblearn) (1.0.2)
Requirement already satisfied: scipy>=1.3.2 in /Users/YannLadouceu
r/opt/anaconda3/lib/python3.9/site-packages (from imbalanced-learn
->imblearn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/Yann
Ladouceur/opt/anaconda3/lib/python3.9/site-packages (from imbalanc
ed-learn->imblearn) (2.2.0)
Requirement already satisfied: joblib>=1.1.1 in /Users/YannLadouce
ur/opt/anaconda3/lib/python3.9/site-packages (from imbalanced-lear
n->imblearn) (1.2.0)
```

In [ ]:
```
# Modélisation - Logistic Regression
```

In [190]:
```
from sklearn.linear_model import LogisticRegression

# Sur-échantillonnage
# Instanciation du modèle
rOs = RandomOverSampler()
# Entrainement du modèle
X_ro, y_ro = rOs.fit_resample(X_train_scaled, y_train)

# Instanciation du modèle
model = LogisticRegression(solver='lbfgs',class_weight='balanced', r
# Entrainement du modèle
model.fit(X_ro, y_ro)

# Affichage de l'accuracy (score) pour les données d'entrainement e
print('Score sur ensemble train', model.score(X_ro, y_ro))
print('Score sur ensemble test', model.score(X_test_scaled, y_test)
```

```
Score sur ensemble train 0.46734252119030395
Score sur ensemble test 0.4148155203733816
```

In [191]:
```
"""
L'analyse du score sur le jeu d'entraînement et sur le jeu de test p
En effet, le score sur le jeu d'entraînement est plus élevé que le s
"""
```

Out[191]:
```
"\nL'analyse du score sur le jeu d'entraînement et sur le jeu de t
est permet d'identifier le surapprentissage.\nEn effet, le score s
ur le jeu d'entraînement est plus élevé que le score de test.\n"
```

In [193]:
```
# Evaluer le modèle en affichant le classification_report_imbalance
from imblearn.metrics import classification_report_imbalanced

# Prédiction
y_pred = model.predict(X_test_scaled)

# Afficher le rapport imbalanced
print(classification_report_imbalanced(y_test, y_pred))

# Matrice de confusion
display(pd.crosstab(y_test, y_pred, rownames = ['Réalité'], colname
```

```
                   pre       rec       spe        f1       geo
iba       sup

          0       0.04      0.49      0.67      0.07      0.57
0.32     16153
          1       0.02      0.51      0.74      0.03      0.61
0.37      5204
          2       0.99      0.41      0.94      0.58      0.62
0.36    566567

avg / total       0.96      0.41      0.93      0.56      0.62
0.36    587924
```

Prédictions	0	1	2
Réalité			
0	7909	7229	1015
1	2191	2643	370
2	186595	146644	233328

In [194]:
```
from sklearn.linear_model import LogisticRegression

# Sous-échantillonnage
# Instanciation du modèle
rUs = RandomUnderSampler()
# Entrainement du modèle
X_ru, y_ru = rUs.fit_resample(X_train_scaled, y_train)

# Instanciation du modèle
model = LogisticRegression(solver='lbfgs',class_weight='balanced', r
# Entrainement du modèle
model.fit(X_ru, y_ru)

# Affichage de l'accuracy (score) pour les données d'entrainement e
print('Score sur ensemble train', model.score(X_ru, y_ru))
print('Score sur ensemble test', model.score(X_test_scaled, y_test)
```

```
Score sur ensemble train 0.47081778806981434
Score sur ensemble test 0.41383580190636887
```

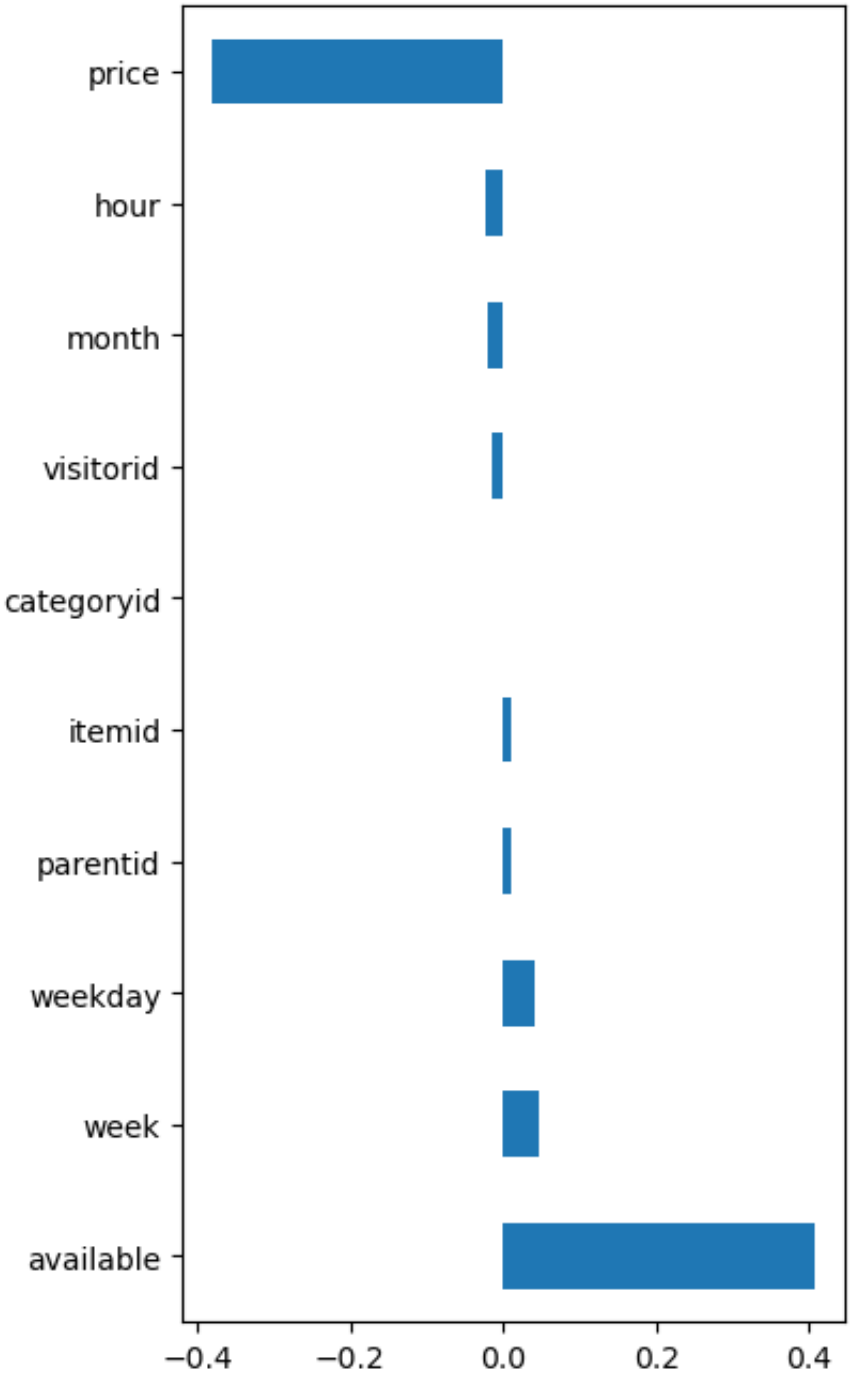
In [195]:
```
# Evaluer le modèle en affichant le classification_report_imbalance
from imblearn.metrics import classification_report_imbalanced

# Prédiction
y_pred = model.predict(X_test_scaled)

# Afficher le rapport imbalanced
print(classification_report_imbalanced(y_test, y_pred))

# Matrice de confusion
display(pd.crosstab(y_test, y_pred, rownames = ['Réalité'], colname
```

```
                   pre       rec       spe        f1       geo
iba       sup

          0       0.04      0.49      0.67      0.07      0.57
0.32     16153
          1       0.02      0.51      0.74      0.03      0.61
0.36      5204
          2       0.99      0.41      0.94      0.58      0.62
0.36    566567

avg / total       0.96      0.41      0.93      0.56      0.62
0.36    587924
```

Prédictions	0	1	2
Réalité			
0	7978	7172	1003
1	2204	2632	368
2	187222	146651	232694

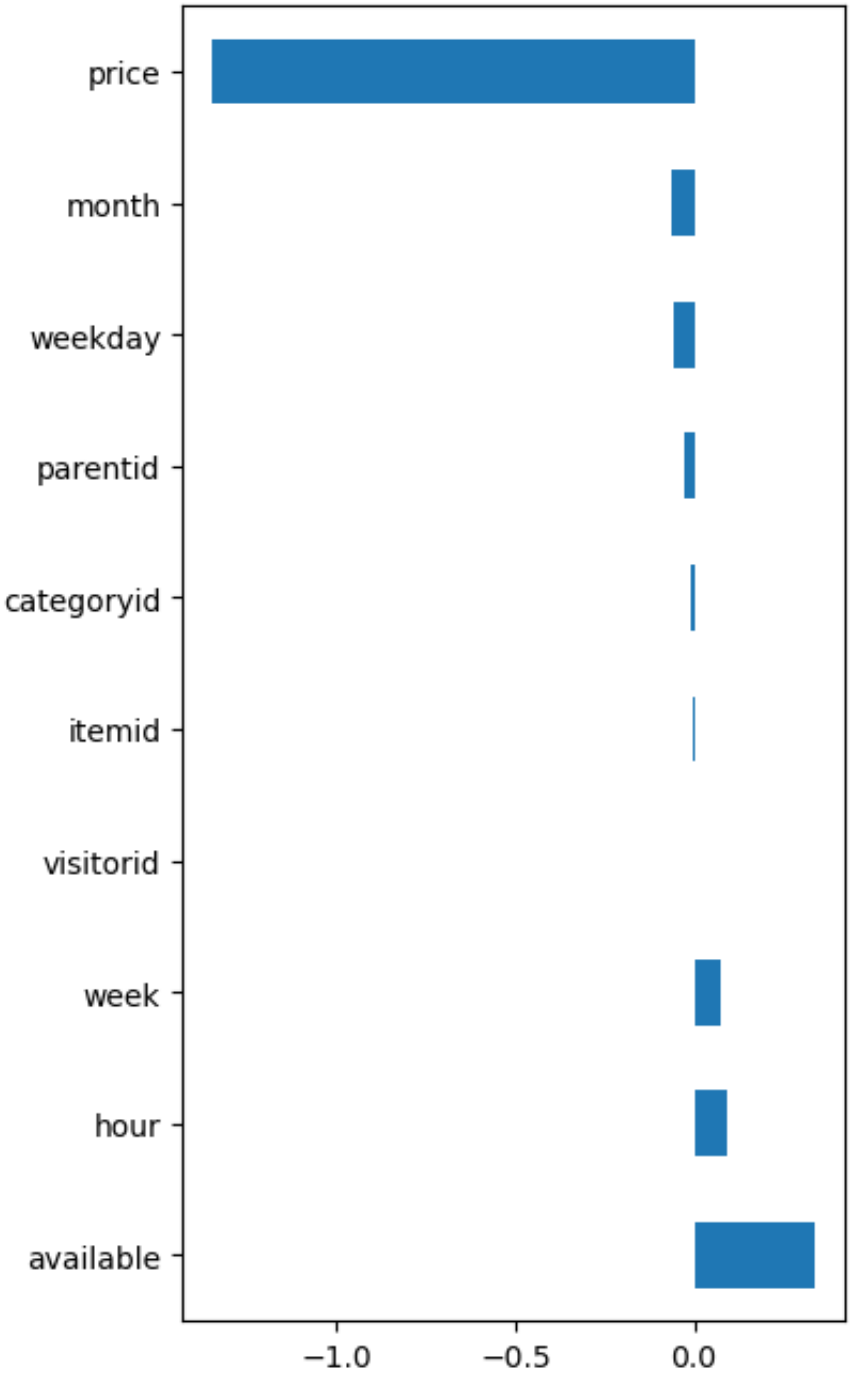
In [198]:
```
# affichage des coefficients associés à chaque variable explicative
display(pd.Series(model.coef_[0], X_train.columns).sort_values(asce
```

```
<AxesSubplot:>
```



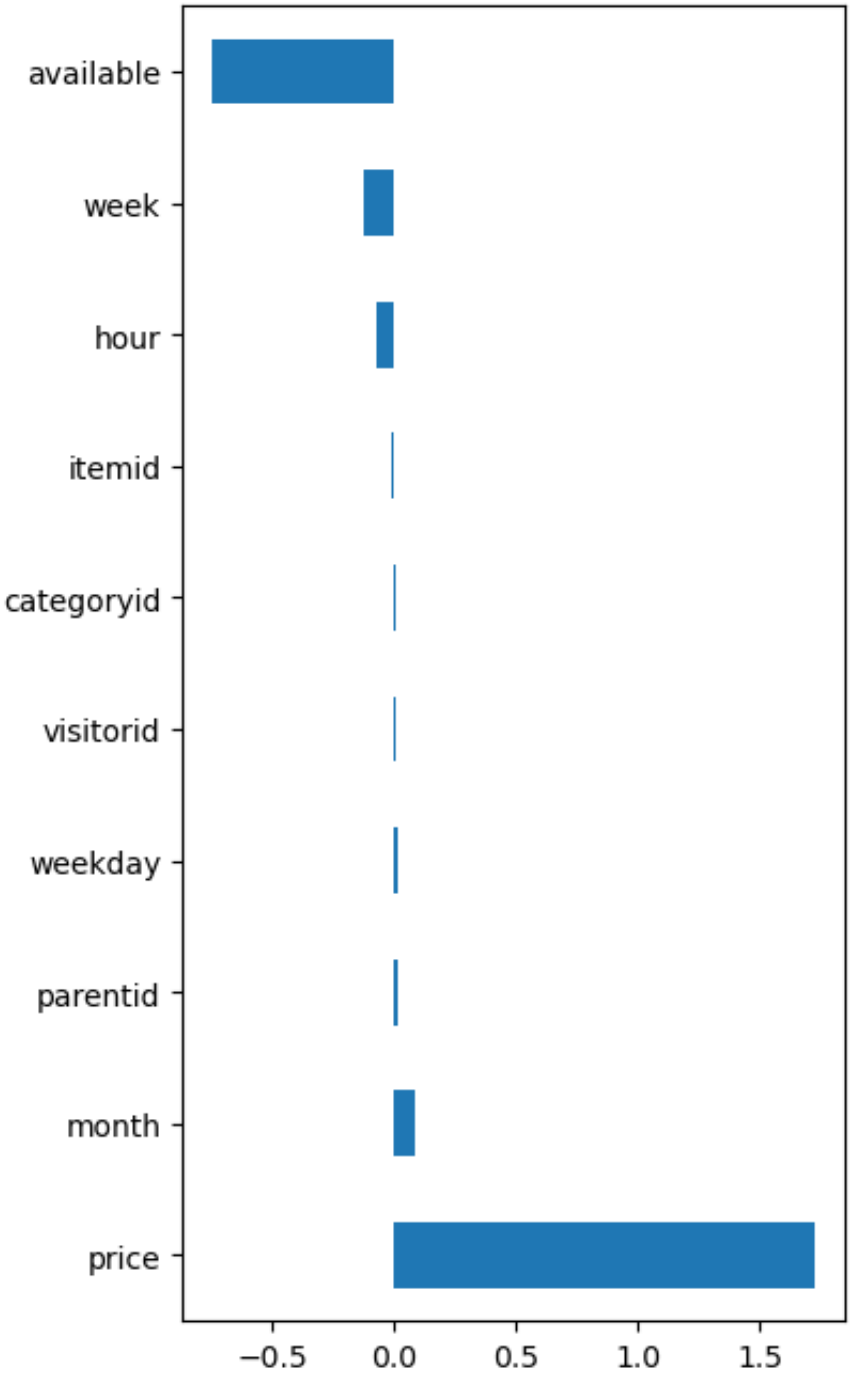
In [199]:
```
display(pd.Series(model.coef_[1], X_train.columns).sort_values(asce
```

```
<AxesSubplot:>
```



In [200]:
```
display(pd.Series(model.coef_[2], X_train.columns).sort_values(asce
```

```
<AxesSubplot:>
```



In [201]:
```
# La transformation des données (normalisation, standardisation, et
"""
Il n'est généralement pas nécessaire d'appliquer une transformation
pour les modèles de forêts aléatoires (Random Forest) ou de gradien
Les modèles de forêts aléatoires et de gradient boosting sont des m
Les arbres de décision fonctionnent en divisant récursivement les d
basés sur les valeurs d'une seule feature à la fois. Ces critères d
de sorte que la normalisation ou la standardisation des données n'e
"""
```

Out[201]:
```
"\nIl n'est généralement pas nécessaire d'appliquer une transforma
tion des données, comme la normalisation ou la standardisation, \n
pour les modèles de forêts aléatoires (Random Forest) ou de gradie
nt boosting, car ces modèles sont généralement peu sensibles à l'é
chelle des données.\nLes modèles de forêts aléatoires et de gradie
nt boosting sont des modèles d'ensemble basés sur des arbres de dé
cision.\nLes arbres de décision fonctionnent en divisant récursive
ment les données en partitions en utilisant des critères de décisi
on \nbasés sur les valeurs d'une seule feature à la fois. Ces crit
ères de décision sont indépendants de l'échelle des données, \nde
sorte que la normalisation ou la standardisation des données n'est
généralement pas nécessaire.\n"
```

In [ ]:
```
# Modélisation - Random Forest
```

In [207]:
```
from sklearn.ensemble import RandomForestClassifier
# Sous-échantillonnage
rUs = RandomUnderSampler()
X_ru, y_ru = rUs.fit_resample(X_train, y_train)

# Instanciation du modèle
rf = RandomForestClassifier(max_depth=3)
# Entrainement sur le modèle RandomForest
rf.fit(X_ru, y_ru)

# Affichage de l'accuracy (score) pour les données d'entrainement e
print('Score sur ensemble train', rf.score(X_ru, y_ru))
print('Score sur ensemble test', rf.score(X_test, y_test))
```

```
Score sur ensemble train 0.4709678802693083
Score sur ensemble test 0.40933011749818005
```

In [208]:
```
"""
L'analyse du score sur le jeu d'entraînement et sur le jeu de test
En effet, le score sur le jeu d'entraînement est plus élevé que le
"""
```

Out[208]:
```
"\nL'analyse du score sur le jeu d'entraînement et sur le jeu de t
est permet d'identifier le surapprentissage.\nEn effet, le score s
ur le jeu d'entraînement est plus élevé que le score de test.\n"
```

In [209]:
```
# Evluer le modèle en affichant le classification_report_imbalanced
from imblearn.metrics import classification_report_imbalanced

# Prédiction
y_pred_rf = rf.predict(X_test)

# Afficher le rapport imbalanced
print(classification_report_imbalanced(y_test, y_pred_rf))

# Matrice de confusion
display(pd.crosstab(y_test, y_pred_rf, rownames = ['Réalité'], coln
```

```
                   pre       rec       spe        f1       geo
iba       sup

          0       0.04      0.48      0.67      0.07      0.57
0.32     16153
          1       0.02      0.54      0.73      0.03      0.63
0.39      5204
          2       0.99      0.41      0.94      0.58      0.62
0.36    566567

avg / total       0.96      0.41      0.93      0.56      0.62
0.36    587924
```

Prédictions	0	1	2
Réalité			
0	7816	7401	936
1	2044	2813	347
2	184746	151795	230026

In [ ]:
```
# Modélisation - GradientBoosting
```

In [212]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Sous-échantillonnage
rUs = RandomUnderSampler()
X_ru, y_ru = rUs.fit_resample(X_train, y_train)

# Instanciation du modèle
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0
# Entrainement du modèle
gb.fit(X_ru, y_ru)

# Affichage de l'accuracy (score) pour les données d'entrainement e
print('Score sur ensemble train', gb.score(X_ru, y_ru))
print('Score sur ensemble test', gb.score(X_test, y_test))
```

```
Score sur ensemble train 0.5879540289034693
Score sur ensemble test 0.5336846259040284
```

In [214]:
```
"""
L'analyse du score sur le jeu d'entraînement et sur le jeu de test
En effet, le score sur le jeu d'entraînement est plus élevé que le
"""
```

Out[214]:
```
"\nL'analyse du score sur le jeu d'entraînement et sur le jeu de t
est permet d'identifier le surapprentissage.\nEn effet, le score s
ur le jeu d'entraînement est plus élevé que le score de test.\n"
```

In [213]:
```
# Evaluer le modèle en affichant le classification_report_imbalance
from imblearn.metrics import classification_report_imbalanced

# Prédiction
y_pred_gb = gb.predict(X_test)

# Afficher le rapport imbalanced
print(classification_report_imbalanced(y_test, y_pred_gb))

# Matrice de confusion
display(pd.crosstab(y_test, y_pred_gb, rownames = ['Réalité'], coln
```

```
                   pre       rec       spe        f1       geo
iba       sup

          0       0.04      0.45      0.73      0.08      0.57
0.32     16153
          1       0.02      0.51      0.80      0.04      0.64
0.39      5204
          2       0.99      0.54      0.84      0.70      0.67
0.43    566567

avg / total       0.95      0.53      0.83      0.67      0.67
0.43    587924
```

Prédictions	0	1	2
Réalité			
0	7286	6095	2772
1	1820	2639	745
2	152933	109793	303841

In [ ]:

In [ ]:

In [215]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils import class_weight

# calcul des poids pour chaque classe en fonction de la distributio
weights = class_weight.compute_class_weight(class_weight = 'balance
class_weights = {i: weights[i] for i in range(len(weights))}

# Initialisation du classifieur
clf = RandomForestClassifier(class_weight=class_weights)

# Entrainement du modèle
clf.fit(X_train, y_train)

# Prédiction sur les données de test
y_pred = clf.predict(X_test)

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

# classification report
print(classification_report(y_test, y_pred))
```

```
[[  1534   3000  11619]
 [  2851    167   2186]
 [ 11140   4786 550641]]
              precision    recall  f1-score   support

           0       0.10      0.09      0.10     16153
           1       0.02      0.03      0.03      5204
           2       0.98      0.97      0.97    566567

    accuracy                           0.94    587924
   macro avg       0.37      0.37      0.37    587924
weighted avg       0.94      0.94      0.94    587924
```

In [216]:
```
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.utils import resample
from sklearn.ensemble import RandomForestClassifier


# Combinaison des jeux de données afin de former un ensemble de don
X = np.concatenate((X_train, X_test))
y = np.concatenate((y_train, y_test))

# Séparation des classes majoritaires et minoritaires pour les moda
#a été encodée)

X_minority_0 = X[y == 0]
y_minority_0 = y[y == 0]
X_majority_0 = X[y != 0]
```

```
y_majority_0 = y[y != 0]

X_minority_1 = X[y == 1]
y_minority_1 = y[y == 1]
X_majority_1 = X[y != 1]
y_majority_1 = y[y != 1]

X_minority_2 = X[y == 2]
y_minority_2 = y[y == 2]
X_majority_2 = X[y != 2]
y_majority_2 = y[y != 2]

# Rééchantillonnage des données par classe
X_minority_upsampled_0, y_minority_upsampled_0 = resample(X_minorit
                                                          replace=Tru
                                                          random_stat

X_minority_upsampled_1, y_minority_upsampled_1 = resample(X_minorit
                                                          replace=Tru
                                                          random_stat

X_minority_upsampled_2, y_minority_upsampled_2 = resample(X_minorit
                                                          replace=Tru
                                                          random_stat

# Concaténation des données de classes minoritaires rééchantillonné
# afin d'obtenir un dataset équilibré
# les données sont à nouveau séparées (x, y)

X_upsampled = np.concatenate((X_majority_0, X_minority_upsampled_0,
y_upsampled = np.concatenate((y_majority_0, y_minority_upsampled_0,

# Entrainement du modèle sur le nouveau dataset. Utilisation d'un R
clf = RandomForestClassifier()
clf.fit(X_upsampled, y_upsampled)

# Prédictions
y_pred = clf.predict(X_test)

# Métriques avec paramètre "weighted"
print("F1 score: ", f1_score(y_test, y_pred, average='weighted'))
print("Precision: ", precision_score(y_test, y_pred, average='weigh
print("Recall: ", recall_score(y_test, y_pred, average='weighted'))
```

```
/Users/YannLadouceur/opt/anaconda3/lib/python3.9/site-packages/skl
earn/base.py:443: UserWarning: X has feature names, but RandomFore
stClassifier was fitted without feature names
  warnings.warn(

F1 score:  0.9679231000603392
Precision:  0.9800664936150993
Recall:  0.9608282703206537
```

In [218]:
```
print("score train : " , clf.score(X_train, y_train))
print("score test : ", clf.score(X_test,y_test))
```

```
/Users/YannLadouceur/opt/anaconda3/lib/python3.9/site-packages/skl
earn/base.py:443: UserWarning: X has feature names, but RandomFore
stClassifier was fitted without feature names
  warnings.warn(

score train :  0.9610873079184405

/Users/YannLadouceur/opt/anaconda3/lib/python3.9/site-packages/skl
earn/base.py:443: UserWarning: X has feature names, but RandomFore
stClassifier was fitted without feature names
  warnings.warn(

score test :  0.9608282703206537
```

In [219]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.49      0.76      0.59     16153
           1       0.34      1.00      0.50      5204
           2       1.00      0.97      0.98    566567

    accuracy                           0.96    587924
   macro avg       0.61      0.91      0.69    587924
weighted avg       0.98      0.96      0.97    587924
```

In [220]:
```
from imblearn.metrics import classification_report_imbalanced
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo
iba       sup

          0       0.49      0.76      0.98      0.59      0.86
0.72     16153
          1       0.34      1.00      0.98      0.50      0.99
0.98      5204
          2       1.00      0.97      1.00      0.98      0.98
0.96    566567

avg / total       0.98      0.96      1.00      0.97      0.98
0.96    587924
```

In [2]:
```
    """
    La métrique GEO (Geometric mean) dans classification_report_imb
    pour les modèles de classification binaire déséquilibrés. Il me
    prenant en compte les taux de faux positifs et de faux négatifs
    géométrique des taux de précision et de rappel, plutôt que la m
    la déséquilibre des classes dans les données d'entraînement, en
    la classe minoritaire.

    """
```

Out[2]:
```
"\nLa métrique GEO (Geometric mean) dans classification_report_imb
alanced est un indicateur de performance \npour les modèles de cla
ssification binaire déséquilibrés. Il mesure la performance de la
classification en \nprenant en compte les taux de faux positifs et
de faux négatifs. Il est calculé en utilisant la moyenne \ngéométr
ique des taux de précision et de rappel, plutôt que la moyenne ari
thmétique. Cela permet de tenir compte de \nla déséquilibre des cl
asses dans les données d'entraînement, en donnant plus de poids au
x éléments mal classés de \nla classe minoritaire.\n    \n"
```

In [ ]:
```
"""
Le F1 score est une métrique de performance couramment utilisée dan
évaluer la précision et le rappel. Il est calculé en utilisant la m
Il est généralement utilisé lorsque les classes sont déséquilibrées
à la fois la précision et le rappel, ce qui est important pour de n

Plus précisément, le F1 score est défini comme :
F1 = 2 * (précision * rappel) / (précision + rappel)
Il varie entre 0 et 1, avec une valeur de 1 indiquant une performan
valeur de 0 indiquant une performance mauvaise (précision et rappel

"""
```

In [1]:
```
from sklearn.inspection import permutation_importance

result = permutation_importance(clf, X_test, y_test, n_repeats=10,
```

```
---------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
/var/folders/s3/78mw79ln4456kxpb8mlqhs8h0000gn/T/ipykernel_94335/1
119726329.py in <module>
      1 from sklearn.inspection import permutation_importance
      2
----> 3 result = permutation_importance(clf, X_test, y_test,
n_repeats=10, random_state=0)

NameError: name 'clf' is not defined
```

In [ ]: