Sommaire

Import des librairies

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as sp
import datetime as dt
import matplotlib.dates as mdates
import seaborn as sns
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

Import des données

In [2]:
```
customers = pd.read_csv('assets/datas/customers.csv')
products = pd.read_csv('assets/datas/products.csv')
transactions = pd.read_csv('assets/datas/transactions.csv',parse_dates=[1])
```

Exploration et nettoyage des données

Customers

In [3]:
```
customers.info()
customers.describe(include="all")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8623 entries, 0 to 8622
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   client_id  8623 non-null   object
 1   sex        8623 non-null   object
 2   birth      8623 non-null   int64 
dtypes: int64(1), object(2)
memory usage: 202.2+ KB
```

Out[3]:

	client_id	sex	birth
count	8623	8623	8623.000000
unique	8623	2	NaN
top	c_4410	f	NaN
freq	1	4491	NaN
mean	NaN	NaN	1978.280877
std	NaN	NaN	16.919535
min	NaN	NaN	1929.000000
25%	NaN	NaN	1966.000000
50%	NaN	NaN	1979.000000
75%	NaN	NaN	1992.000000
max	NaN	NaN	2004.000000

In [4]:
```
# Doublons de customers
customers.isnull().sum()
```

Out[4]:
```
client_id    0
sex          0
birth        0
dtype: int64
```

In [5]:
```
print("Nombre doublon de customers['client_id'] : ",customers['client_id'].duplicate
```

```
Nombre doublon de customers['client_id'] :  0
```

In [6]:
```
# Vérification des erreurs de saisies dans customers[sex]
customers['sex'].unique()
```

Out[6]:
```
array(['f', 'm'], dtype=object)
```

In [7]:
```
# Création de la colonne age
from datetime import datetime
current_year = datetime.now().strftime('%Y')
customers['age'] = int(current_year)-customers['birth']
customers.head()
```

Out[7]:

	client_id	sex	birth	age
0	c_4410	f	1967	55
1	c_7839	f	1975	47
2	c_1699	f	1984	38
3	c_5961	f	1962	60
4	c_5320	m	1943	79

In [8]:
```
customers.describe()
```

Out[8]:

	birth	age
count	8623.000000	8623.000000
mean	1978.280877	43.719123
std	16.919535	16.919535
min	1929.000000	18.000000
25%	1966.000000	30.000000
50%	1979.000000	43.000000
75%	1992.000000	56.000000
max	2004.000000	93.000000

In [9]:
```
print("Customers :")
print('- Le df customers est propre')
print('- les types des données sont bons')
print("- l'ID est 'client_id'")
print("- Les valeurs de sex sont m pour les hommes et f pour les femmes")
```

```
print("- il y a : ",customers.shape[0],"lignes")
print("-----")
print("- Le plus jeune acheteur est agé de 18 ans")
print("- Le plus agé est agé de 93 ans")
```

```
Customers :
- Le df customers est propre
- les types des données sont bons
- l'ID est 'client_id'
- Les valeurs de sex sont m pour les hommes et f pour les femmes
- il y a :  8623 lignes
-----
- Le plus jeune acheteur est agé de 18 ans
- Le plus agé est agé de 93 ans
```

Products

In [10]:
```
products.info()
products.describe(include='all')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3287 entries, 0 to 3286
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   id_prod  3287 non-null   object 
 1   price    3287 non-null   float64
 2   categ    3287 non-null   int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 77.2+ KB
```

Out[10]:

	id_prod	price	categ
count	3287	3287.000000	3287.000000
unique	3287	NaN	NaN
top	0_1421	NaN	NaN
freq	1	NaN	NaN
mean	NaN	21.856641	0.370246
std	NaN	29.847908	0.615387
min	NaN	-1.000000	0.000000
25%	NaN	6.990000	0.000000
50%	NaN	13.060000	0.000000
75%	NaN	22.990000	1.000000
max	NaN	300.000000	2.000000

Le prix minimum est : -1, c'est incohérent on identifie et supprimes les lignes correspondantes

In [11]:
```
products.loc[products['price']<0,:]
```

Out[11]:

	id_prod	price	categ
731	T_0	-1.0	0

In [12]:
```
# Suppression des lignes de tests
products = products[products.id_prod !='T_0']
products.shape
```

Out[12]:
```
(3286, 3)
```

In [13]:
```
products.isnull().sum()
```

Out[13]:
```
id_prod    0
price      0
categ      0
dtype: int64
```

In [14]:
```
print("products : ")
print("le df products est propre")
print("il y a ", products.shape[0],"produits vendus par l'entreprise")
```

```
products :
le df products est propre
il y a  3286 produits vendus par l'entreprise
```

transactions

In [15]:
```
transactions.info()
transactions.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 679532 entries, 0 to 679531
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id_prod     679532 non-null  object
 1   date        679532 non-null  object
 2   session_id  679532 non-null  object
 3   client_id   679532 non-null  object
dtypes: object(4)
memory usage: 20.7+ MB
```

Out[15]:

	id_prod	date	session_id	client_id
count	679532	679532	679532	679532
unique	3267	679371	342316	8602
top	1_369	test_2021-03-01 02:30:02.237413	s_0	c_1609
freq	2252	13	200	25488

In [16]:
```
# Question de conversion de la colonne date
print("Ici On ne peut pas convertir la colonne date dans le bon type en raison de ce
```

```
Ici On ne peut pas convertir la colonne date dans le bon type en raison de certaines
cellules. Il faut :
- identifier les lignes concernées correspondants à des lignes de test
- définir leur utilité dans le df
- les supprimer si elle n'apportent rien
```

In [17]:
```
# identifier les lignes de test
transactions.loc[transactions['session_id']=='s_0',:]
```

Out[17]:

	id_prod	date	session_id	client_id
3019	T_0	test_2021-03-01 02:30:02.237419	s_0	ct_0
5138	T_0	test_2021-03-01 02:30:02.237425	s_0	ct_0
9668	T_0	test_2021-03-01 02:30:02.237437	s_0	ct_1
10728	T_0	test_2021-03-01 02:30:02.237436	s_0	ct_0
15292	T_0	test_2021-03-01 02:30:02.237430	s_0	ct_0
...	...	...	...	...
657830	T_0	test_2021-03-01 02:30:02.237417	s_0	ct_0
662081	T_0	test_2021-03-01 02:30:02.237427	s_0	ct_1
670680	T_0	test_2021-03-01 02:30:02.237449	s_0	ct_1
671647	T_0	test_2021-03-01 02:30:02.237424	s_0	ct_1
679180	T_0	test_2021-03-01 02:30:02.237425	s_0	ct_1

200 rows × 4 columns

Nettoyage du DF transaction et reprise du df customers

In [18]:
```
print("Les lignes de test ont toutes le même id : 'T_0' qui correspond à la suppress
print("On supprime aussi les clients de client_id: 'ct_1', 'ct_0' du df customers")
```

```
Les lignes de test ont toutes le même id : 'T_0' qui correspond à la suppression de
200 lignes
On supprime aussi les clients de client_id: 'ct_1', 'ct_0' du df customers
```

In [19]:
```
#Suppression des lignes de test dan transaction
transactions=transactions[transactions.id_prod != 'T_0'];
customers.drop(transactions[(transactions["client_id"] == "ct_0")].index, inplace=Tr
customers.drop(transactions[(transactions["client_id"] == "ct_1")].index, inplace=Tr
#Suppression des éléments de test dans customers
customers.drop(customers[(customers["client_id"] == "ct_0")].index, inplace=True)
customers.drop(customers[(customers["client_id"] == "ct_1")].index, inplace=True)
```

In [20]:
```
#Conversion de la colonne date dans le bon type
print("Conversion de la colonne date en datetime")
transactions['date'] = pd.to_datetime(transactions['date'])
transactions.dtypes
```

Out[20]:
```
Conversion de la colonne date en datetime
id_prod               object
date          datetime64[ns]
session_id            object
client_id             object
dtype: object
```

In [21]:
```
print("Nombre de clients uniques : ",len(transactions['client_id'].unique()),"client
```

```
Nombre de clients uniques :  8600 clients
```

Les df sont nettoyés

Jointures

Unicité des clés

In [22]:
```
print("df : customers : Nombre des 'client_id' uniques",customers['client_id'].nuniq
print("df : products : Nombre des 'prod_id' uniques",products['id_prod'].nunique(),"
```

```
df : customers : Nombre des 'client_id' uniques 8621  pour  8621 lignes
df : products : Nombre des 'prod_id' uniques 3286  pour  3286 lignes
```

Les clés primaires :

- Pour le dataframe **customers 'client_id'** apparait comme un bonne clé : 8621 client_id unique pour 8621 clients uniques
- Pour le dataframe **products 'id_prod'** apparaît comme un bonne clé :3286 id_prod unique pour 3286 produits vendus par l'entreprise

DF : jointure globale

On utilise outer pour conserver toutes les lignes

In [23]:
```
df = pd.merge(transactions, products, on='id_prod', how='outer')
df = pd.merge(df,customers, on="client_id",how='outer')
df.head()
```

Out[23]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age
0	0_1518	2022-05-20 13:21:29.043970	s_211425	c_103	4.18	0.0	f	1986.0	36.0
1	0_1518	2021-07-20 13:21:29.043970	s_64849	c_103	4.18	0.0	f	1986.0	36.0
2	0_1518	2022-08-20 13:21:29.043970	s_255965	c_103	4.18	0.0	f	1986.0	36.0
3	0_1418	2022-06-18 01:49:37.823274	s_225411	c_103	8.57	0.0	f	1986.0	36.0
4	0_1418	2021-08-18 01:49:37.823274	s_77214	c_103	8.57	0.0	f	1986.0	36.0

In [24]:
```
print(df.isna().mean()*100)
```

```
id_prod       0.003091
date          0.006182
session_id    0.006182
client_id     0.003091
price         0.035621
categ         0.035621
sex           0.003091
birth         0.003091
age           0.003091
dtype: float64
```

In [25]:
```
print("Après les jointures il y a ",df.shape[0],"lignes et ",df.shape[1],"colonnes")
```

```
Après les jointures il y a  679374 lignes et  9 colonnes
```

In [26]:
```
#Création des colonnes years,month, day pour faciliter la manipulation des données
df['years']=df['date'].dt.year
```

```
df['month']=df['date'].dt.month
df['day']=df['date'].dt.day
df.dtypes
```

Out[26]:
```
id_prod               object
date          datetime64[ns]
session_id            object
client_id             object
price                float64
categ                float64
sex                   object
birth                float64
age                  float64
years                float64
month                float64
day                  float64
dtype: object
```

In [27]:
```
# Vérifications de la saisie des dates
print("Nombre d'années : ",df["years"].nunique(),"\nNombres de mois : ",df["month"].
```

```
Nombre d'années :  3
Nombres de mois :  12
Nombre de jour :  31
Pas d'erreur de saisie
```

In [28]:
```
# Nombre de données manquantes dans le df
df.isnull().mean()*100
```

Out[28]:
```
id_prod       0.003091
date          0.006182
session_id    0.006182
client_id     0.003091
price         0.035621
categ         0.035621
sex           0.003091
birth         0.003091
age           0.003091
years         0.006182
month         0.006182
day           0.006182
dtype: float64
```

In [29]:
```
df.isnull().sum()
```

Out[29]:
```
id_prod        21
date           42
session_id     42
client_id      21
price         242
categ         242
sex            21
birth          21
age            21
years          42
month          42
day            42
dtype: int64
```

In [30]:
```
df.loc[df['price'].isnull(),:]
```

Out[30]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	years	mon
37861	0_2245	2021-04-22 04:57:20.090378	s_23987	c_6714	NaN	NaN	f	1968.0	54.0	2021.0	4
37862	0_2245	2021-06-05 17:04:43.982913	s_44481	c_6714	NaN	NaN	f	1968.0	54.0	2021.0	6
37863	0_2245	2022-04-05 17:04:43.982913	s_189669	c_6714	NaN	NaN	f	1968.0	54.0	2022.0	4
37864	0_2245	2022-05-05 17:04:43.982913	s_204093	c_6714	NaN	NaN	f	1968.0	54.0	2022.0	5
37865	0_2245	2022-12-05 17:04:43.982913	s_307520	c_6714	NaN	NaN	f	1968.0	54.0	2022.0	12
...	...	...	...	...	...	...	...	...	...	...	
679369	NaN	NaT	NaN	c_862	NaN	NaN	f	1956.0	66.0	NaN	Na
679370	NaN	NaT	NaN	c_7584	NaN	NaN	f	1960.0	62.0	NaN	Na
679371	NaN	NaT	NaN	c_90	NaN	NaN	m	2001.0	21.0	NaN	Na
679372	NaN	NaT	NaN	c_587	NaN	NaN	m	1993.0	29.0	NaN	Na
679373	NaN	NaT	NaN	c_3526	NaN	NaN	m	1956.0	66.0	NaN	Na

242 rows × 12 columns

In [31]:
```
df.loc[df['id_prod']=='0_2245',:]
```

Out[31]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	years	mon
37861	0_2245	2021-04-22 04:57:20.090378	s_23987	c_6714	NaN	NaN	f	1968.0	54.0	2021.0	4
37862	0_2245	2021-06-05 17:04:43.982913	s_44481	c_6714	NaN	NaN	f	1968.0	54.0	2021.0	6
37863	0_2245	2022-04-05 17:04:43.982913	s_189669	c_6714	NaN	NaN	f	1968.0	54.0	2022.0	4
37864	0_2245	2022-05-05 17:04:43.982913	s_204093	c_6714	NaN	NaN	f	1968.0	54.0	2022.0	5
37865	0_2245	2022-12-05 17:04:43.982913	s_307520	c_6714	NaN	NaN	f	1968.0	54.0	2022.0	12
...	...	...	...	...	...	...	...	...	...	...	
654934	0_2245	2021-09-07 20:55:19.719028	s_86505	c_8153	NaN	NaN	f	1975.0	47.0	2021.0	9
656406	0_2245	2021-10-20 13:11:05.671456	s_107564	c_1746	NaN	NaN	m	1994.0	28.0	2021.0	10
660365	0_2245	2021-04-10 06:15:32.619826	s_18510	c_277	NaN	NaN	f	2000.0	22.0	2021.0	4
660366	0_2245	2022-04-10 06:15:32.619826	s_191872	c_277	NaN	NaN	f	2000.0	22.0	2022.0	4
660514	0_2245	2021-10-15 09:31:31.539354	s_105069	c_4188	NaN	NaN	f	1935.0	87.0	2021.0	10

221 rows × 12 columns

In [32]:
```
# Remplacement des données manquantes du produit 0_2245 par le prix median accentue
df["categ"] = np.where(df["id_prod"] == "0_2245",0,df["categ"])
df["price"] = np.where(df["id_prod"] == "0_2245",df.loc[(df["categ"]==0),"price"].me
df.isna().sum()
```

Out[32]:
```
id_prod       21
date          42
session_id    42
client_id     21
price         21
categ         21
sex           21
birth         21
age           21
years         42
month         42
day           42
dtype: int64
```

In [33]:
```
#Liste des produits sans identifiant de l'id 679353 à l'id 679973
df.loc[df['id_prod'].isnull()==True,:]
```

Out[33]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	years	month	day
679353	NaN	NaT	NaN	c_8253	NaN	NaN	f	2001.0	21.0	NaN	NaN	NaN
679354	NaN	NaT	NaN	c_3789	NaN	NaN	f	1997.0	25.0	NaN	NaN	NaN
679355	NaN	NaT	NaN	c_4406	NaN	NaN	f	1998.0	24.0	NaN	NaN	NaN
679356	NaN	NaT	NaN	c_2706	NaN	NaN	f	1967.0	55.0	NaN	NaN	NaN
679357	NaN	NaT	NaN	c_3443	NaN	NaN	m	1959.0	63.0	NaN	NaN	NaN
679358	NaN	NaT	NaN	c_4447	NaN	NaN	m	1956.0	66.0	NaN	NaN	NaN
679359	NaN	NaT	NaN	c_3017	NaN	NaN	f	1992.0	30.0	NaN	NaN	NaN
679360	NaN	NaT	NaN	c_4086	NaN	NaN	f	1992.0	30.0	NaN	NaN	NaN
679361	NaN	NaT	NaN	c_6930	NaN	NaN	m	2004.0	18.0	NaN	NaN	NaN
679362	NaN	NaT	NaN	c_4358	NaN	NaN	m	1999.0	23.0	NaN	NaN	NaN
679363	NaN	NaT	NaN	c_8381	NaN	NaN	f	1965.0	57.0	NaN	NaN	NaN
679364	NaN	NaT	NaN	c_1223	NaN	NaN	m	1963.0	59.0	NaN	NaN	NaN
679365	NaN	NaT	NaN	c_6862	NaN	NaN	f	2002.0	20.0	NaN	NaN	NaN
679366	NaN	NaT	NaN	c_5245	NaN	NaN	f	2004.0	18.0	NaN	NaN	NaN
679367	NaN	NaT	NaN	c_5223	NaN	NaN	m	2003.0	19.0	NaN	NaN	NaN
679368	NaN	NaT	NaN	c_6735	NaN	NaN	m	2004.0	18.0	NaN	NaN	NaN
679369	NaN	NaT	NaN	c_862	NaN	NaN	f	1956.0	66.0	NaN	NaN	NaN
679370	NaN	NaT	NaN	c_7584	NaN	NaN	f	1960.0	62.0	NaN	NaN	NaN
679371	NaN	NaT	NaN	c_90	NaN	NaN	m	2001.0	21.0	NaN	NaN	NaN
679372	NaN	NaT	NaN	c_587	NaN	NaN	m	1993.0	29.0	NaN	NaN	NaN

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	years	month	day
679373	NaN	NaT	NaN	c_3526	NaN	NaN	m	1956.0	66.0	NaN	NaN	NaN

Suite aux jointures :

- Le choix de conserver toutes les lignes permet l'émergence de NaN que nous pourront analyser par la suite :
  - 21 clients ont un compte sur le site et n'ont pas fait d'achats
  - 42 sessions n'ont pas aboutis à un achats
  - Le produit '0_2245' apparaît mais ne dispose pas de prix ni de catégorie, soit le produit a été supprimer soit c'est une erreur mais je n'ai pas l'information. Il représente 221 NaN sur 242
- Pour faire des calculs sur les prix et les valeurs, nous faisons un autre df nommer achats pour supprimer sans NaN

DF : jointure achats

Pour les calculs on crée le df achats

- On utilise inner pour conserver toutes les lignes

In [34]:
```
achats = df.dropna()
```

In [35]:
```
print(achats.isna().mean()*100)
```

```
id_prod       0.0
date          0.0
session_id    0.0
client_id     0.0
price         0.0
categ         0.0
sex           0.0
birth         0.0
age           0.0
years         0.0
month         0.0
day           0.0
dtype: float64
```

In [36]:
```
achats.duplicated().count()
```

Out[36]:
```
679332
```

In [37]:
```
achats['years']=achats['date'].dt.year
achats['month']=achats['date'].dt.month
achats['day']=achats['date'].dt.day
achats.dtypes
```

```
C:\Users\tomy\AppData\Local\Temp/ipykernel_10484/2144409395.py:1: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  achats['years']=achats['date'].dt.year
C:\Users\tomy\AppData\Local\Temp/ipykernel_10484/2144409395.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  achats['month']=achats['date'].dt.month
C:\Users\tomy\AppData\Local\Temp/ipykernel_10484/2144409395.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  achats['day']=achats['date'].dt.day
```

Out[37]:

```
id_prod              object
date         datetime64[ns]
session_id           object
client_id            object
price               float64
categ               float64
sex                  object
birth               float64
age                 float64
years                 int64
month                 int64
day                   int64
dtype: object
```

In [38]:

```
print("Le df achats dispose de",achats.shape[0],"lignes et",achats.shape[1],"colonne
```

```
Le df achats dispose de 679332 lignes et 12 colonnes
```

Analyse du chiffre d'affaire

Différents indicateurs et graphiques autour du chiffre d'affaires et évolution dans le temps

In [39]:

```
# calcul de ca par la somme des prix de la colonne price de df
print("Chiffre d'affaire par la somme des price de DF : ",round(df.price.sum(),2),"€
print("Vérification du Chiffre d'affaire total        : ",round(df.loc[(df['session_
CA = round(df.price.sum(),2)
```

```
Chiffre d'affaire par la somme des price de DF :  11856731.75 €
Vérification du Chiffre d'affaire total        :  11856731.75 €
```

In [40]:

```
# nombre de vente et vérifications
nb_achats = achats['session_id'].count()
print("Nombre de produit vendus par le df achats : ",achats['session_id'].count())
print("Nombre de produit vendus par le df df: ",df['session_id'].count())
print("Difference entre les 2 df : ",df['session_id'].count()-achats['session_id'].c
```

```
Nombre de produit vendus par le df achats :  679332
Nombre de produit vendus par le df df:  679332
Difference entre les 2 df :  0
```

In [41]:
```
# Calcul du CA total
print("Le Chiffre d 'affaire total est de",achats.price.sum(),"euros, pour",achats['
# Nombre totale de ventes
nb_vente = achats['date'].count()
print("Nombre de ventes au total : ",nb_vente,"ventes")
```

```
Le Chiffre d 'affaire total est de 11855936.47 euros, pour 679332 ventes
Nombre de ventes au total :  679332 ventes
```

In [42]:
```
# Chiffres d'affaires annuel par année
print("Ventes annuelles en €")
print(achats.groupby(['years'])['price'].sum())

print('\n')

# Nombres de ventes annuel
nb_ventes_annuel = achats.groupby(achats['years'])['session_id'].count()
print("Nombre de vente",nb_ventes_annuel)
```

```
Ventes annuelles en €
years
2021    4771695.69
2022    6109880.61
2023     974360.17
Name: price, dtype: float64


Nombre de vente years
2021    278335
2022    346500
2023     54497
Name: session_id, dtype: int64
```

In [43]:
```
print(df["date"].nlargest(1))
print(df["date"].nsmallest(1))
```

```
398652   2023-02-28 23:58:30.792755
Name: date, dtype: datetime64[ns]
468405   2021-03-01 00:01:07.843138
Name: date, dtype: datetime64[ns]
```

In [44]:
```
print("La meilleure année est 2022 en chiffre d'affaire et nombre de ventes")
print("Les données de 2023 sont imcomplètes, la dernière opération enregistrée est d
df['date'].nlargest(1)
```

```
La meilleure année est 2022 en chiffre d'affaire et nombre de ventes
Les données de 2023 sont imcomplètes, la dernière opération enregistrée est daté du
28 février 2023 :
```

Out[44]:
```
398652   2023-02-28 23:58:30.792755
Name: date, dtype: datetime64[ns]
```

In [45]:
```
# taux de croissance
ca_annuel = achats.groupby(['years'])['price'].sum()
print(ca_annuel.index)
print ("Entre 2021 et 2022 le chiffre d'affaire a connu une progression de",round((c
print ("Entre 2022 et 2023 le chiffre d'affaire a connu une progression de",round((c
```

```
Int64Index([2021, 2022, 2023], dtype='int64', name='years')
Entre 2021 et 2022 le chiffre d'affaire a connu une progression de 28.04 %
Entre 2022 et 2023 le chiffre d'affaire a connu une progression de -84.05 %
```

In [46]:

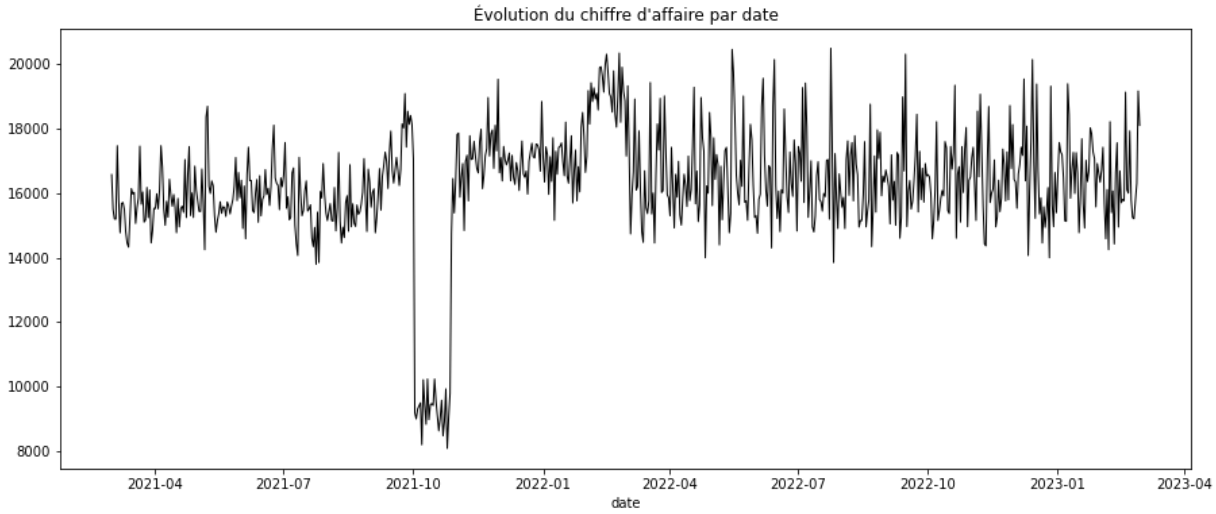
```
# Les mesures de tendances centrale du tableau achats mode,median, mean
print("moy:\n",achats['price'].mean())
print("med:\n",achats['price'].median())
print("mod:\n",achats['price'].mode())
```

```
moy:
 17.452345053759284
med:
 13.99
mod:
 0    15.99
dtype: float64
```

Visualisation du CA

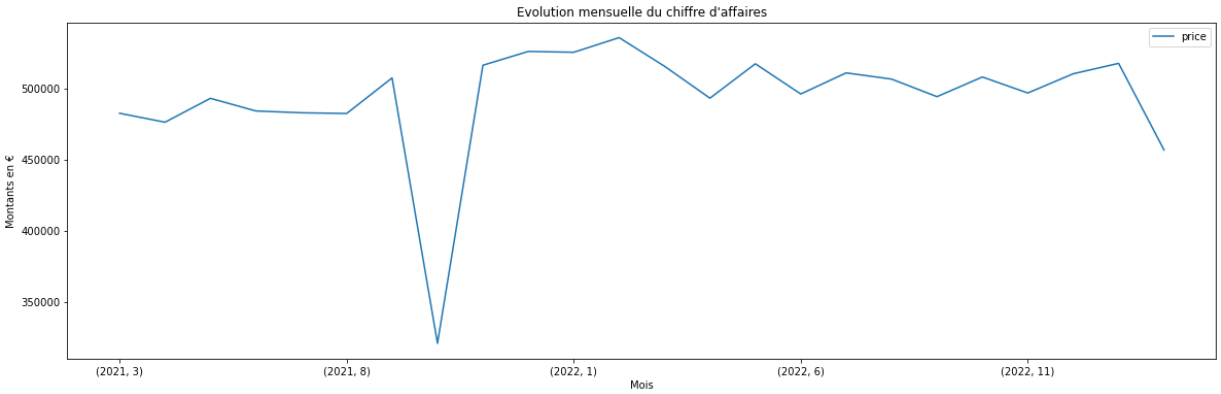
In [47]:

```
fig = plt.figure(figsize=(15,6))
plt.title("Évolution du chiffre d'affaire par date")
achats.groupby(achats['date'].dt.date).sum()['price'].plot(color="black",lw=1)
fig.savefig("assets/graphiques/1_Evolution_du_CA_par_date.png")
```



In [48]:

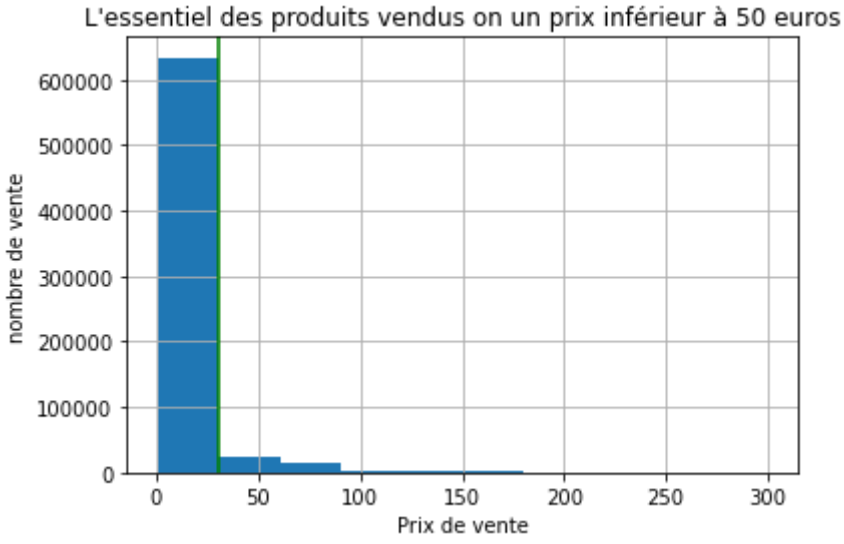
```
#Visualisation par mois
# ca_annnuel
achats = achats.sort_values(["date"],ascending=True)
ca_annuel = achats.groupby(["years"])["price"].sum()

#ca_par_mois
achats = achats.sort_values(["date"],ascending=True)
ca_par_mois = achats.groupby(['years','month'])['price'].sum()

#Graphique par mois
fig= plt.figure(figsize = (20,6))
ca_par_mois.plot()
plt.xlabel("Mois")
plt.ylabel("Montants en €")
plt.title("Evolution mensuelle du chiffre d'affaires")
plt.legend()
fig.savefig("assets/graphiques/2_Evolution_du_CA_par_mois.png")
```

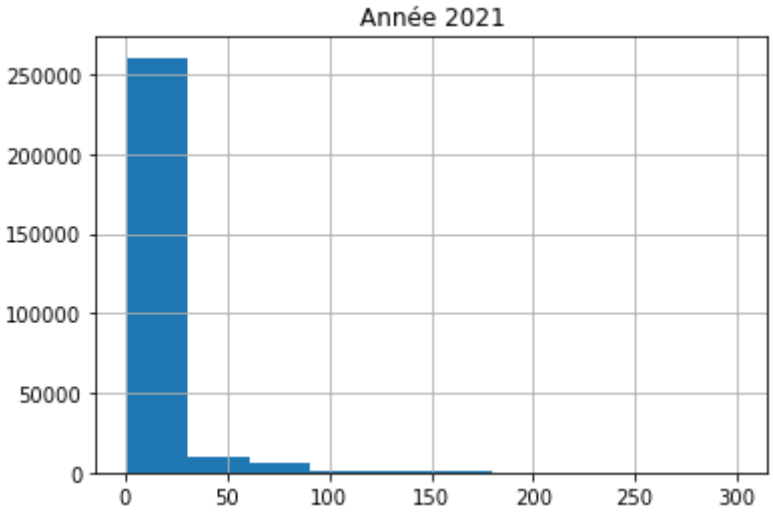


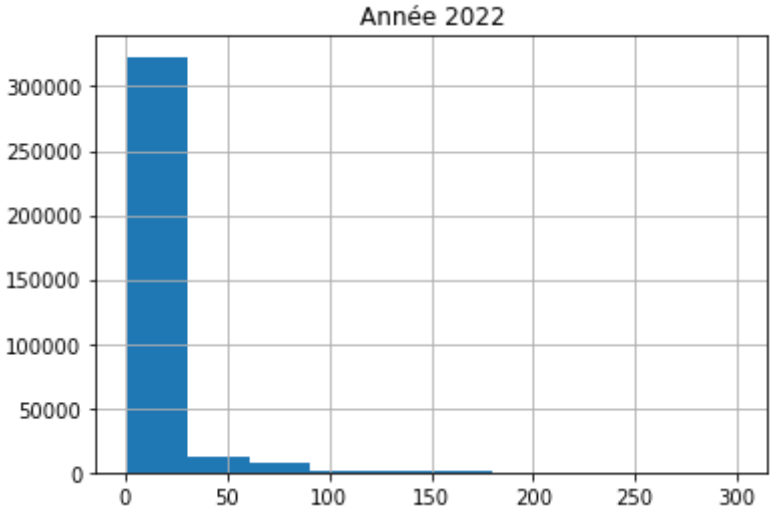
In [49]:

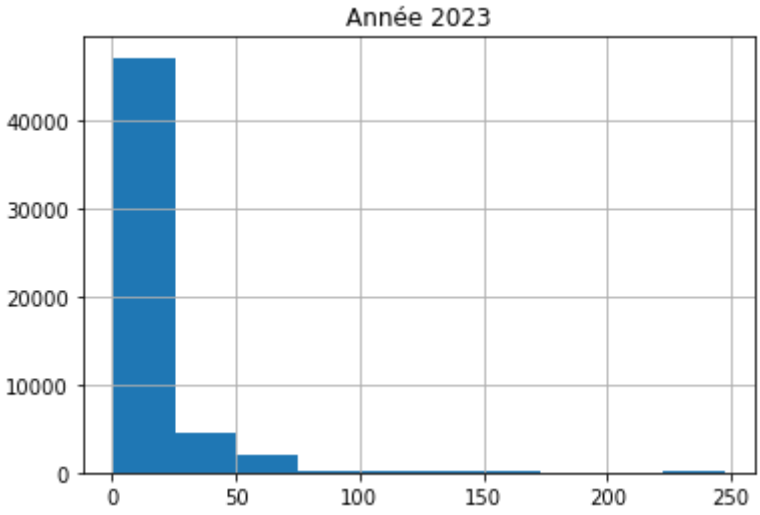
```
#Visulisation de la distribution des prix
print("Représentation des la distribution empirique")
df['price'].hist(density=False)
plt.title("L'essentiel des produits vendus on un prix inférieur à 50 euros")
plt.ylabel('nombre de vente')
plt.xlabel('Prix de vente')
plt.axvline(x=30,color='green')
plt.show()

df.loc[df['years']==2021]["price"].hist(density=False)
plt.title("Année 2021")
plt.show()

df.loc[df['years']==2022]["price"].hist(density=False)
plt.title("Année 2022")
plt.show()

df.loc[df['years']==2023]["price"].hist(density=False)
plt.title("Année 2023")
plt.show()
```

```
Représentation des la distribution empirique
```









Paniers Moyens

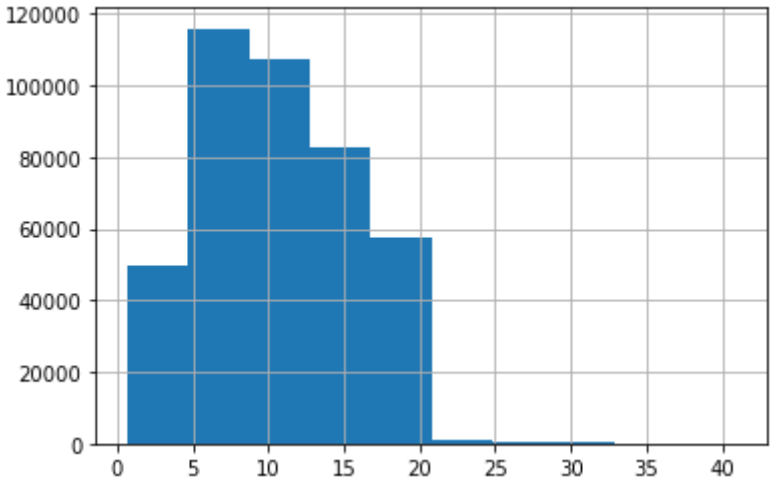
Analyse par catégorie

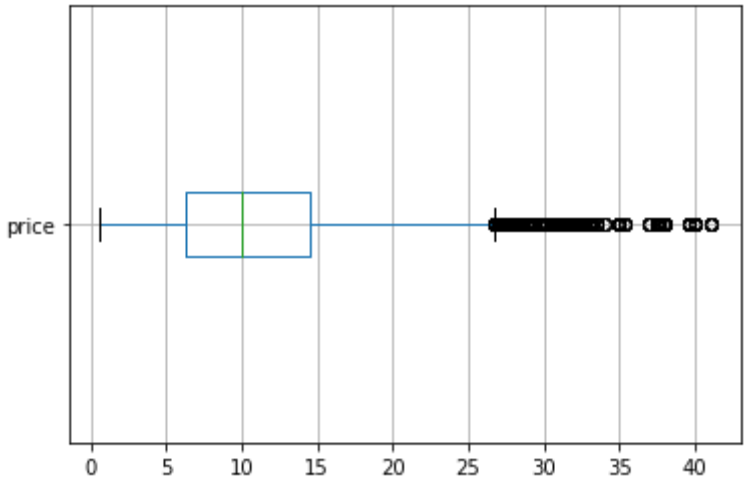
In [129…

```
# MIEUX avec mesures de dispersions
for cat in achats["categ"].unique():
    subset = achats[achats.categ == cat]
    print("-"*20)
    print(cat)
    print("moy:\n",subset['price'].mean())
```

```
    print("med:\n",subset['price'].median())
    print("mod:\n",subset['price'].mode())
    print("var:\n",subset['price'].var(ddof=0))
    print("ect:\n",subset['price'].std(ddof=0))
    print("skw:\n",subset['price'].skew())
    print("kur:\n",subset['price'].kurtosis())
    subset["price"].hist()
    plt.show()
    subset.boxplot(column="price", vert=False)
    plt.show()
```

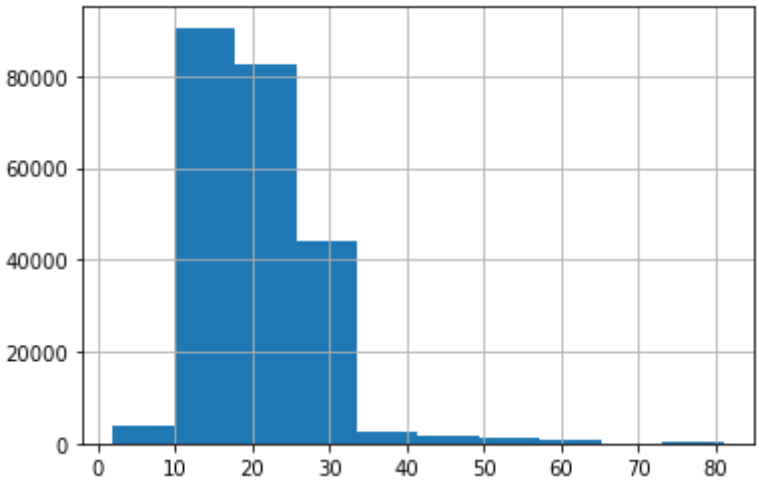
```
--------------------
0.0
moy:
 10.63784343731548
med:
 9.99
mod:
 0    4.99
dtype: float64
var:
 24.32691596173827
ect:
 4.932232350745275
skw:
 0.42268123165343996
kur:
 -0.38609492177615223
```

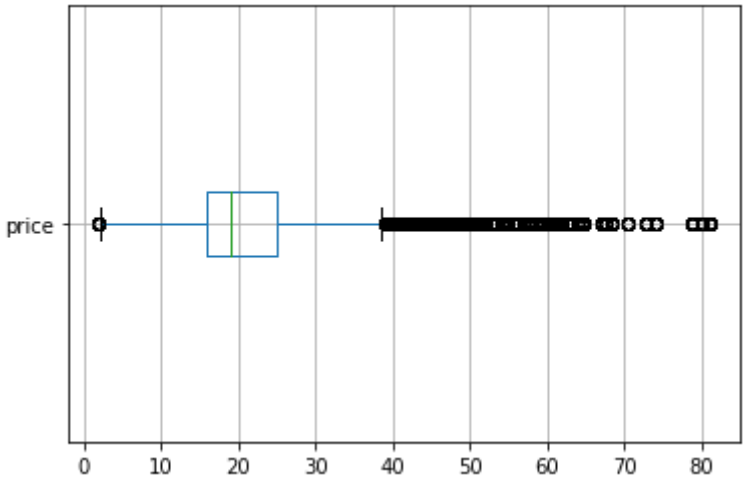




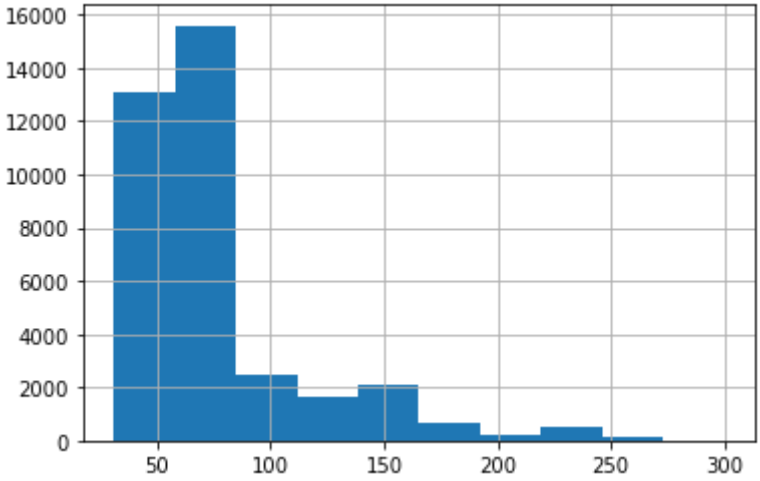

```
--------------------
1.0
moy:
 20.4857295230179
med:
```

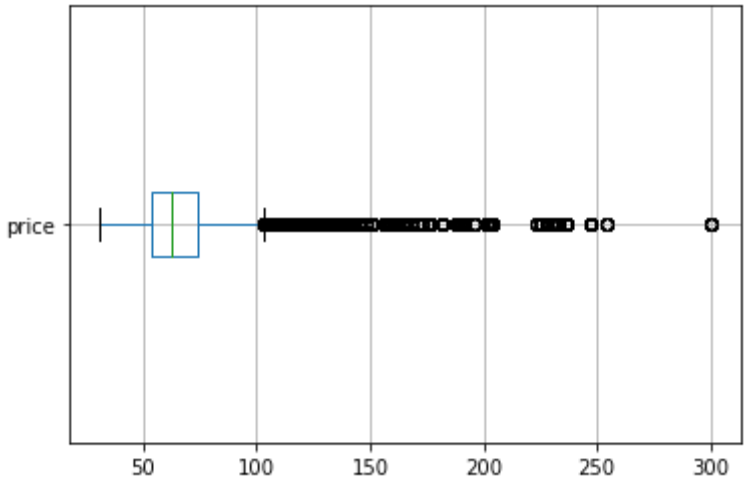
```
 19.08
mod:
 0    15.99
dtype: float64
var:
 57.53035707569349
ect:
 7.584876866218297
skw:
 1.734237258116213
kur:
 8.164115605848043
```






```
--------------------
2.0
moy:
 76.20741221941681
med:
 62.83
mod:
 0    68.99
dtype: float64
var:
 1579.9408534984402
ect:
 39.74846982587431
skw:
 2.1835172974115067
kur:
 4.817061548670198
```

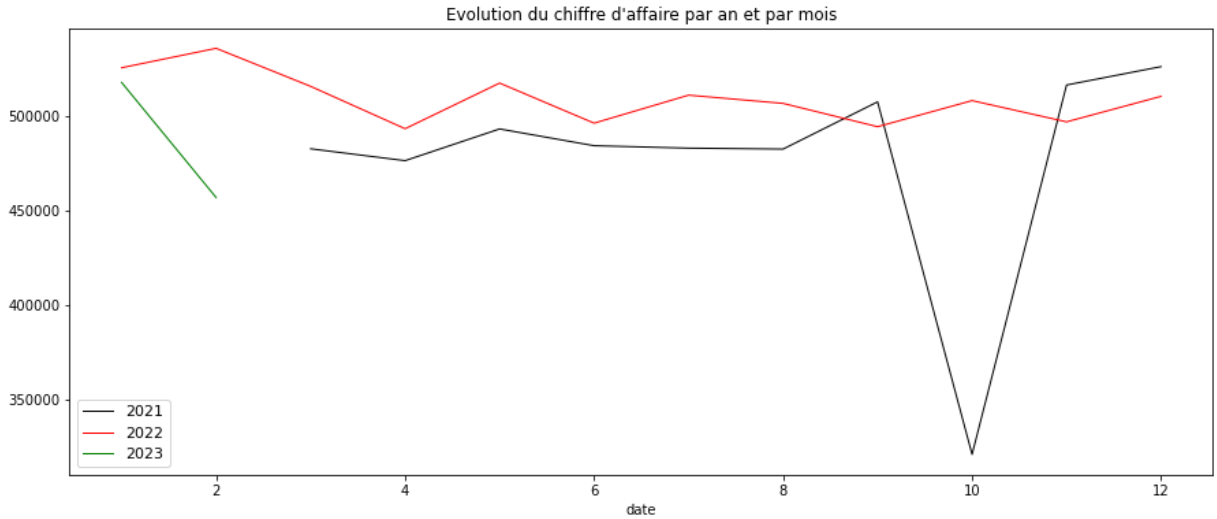




Baisse brutale du chiffre d'affaire en octobre 2021

In [52]:

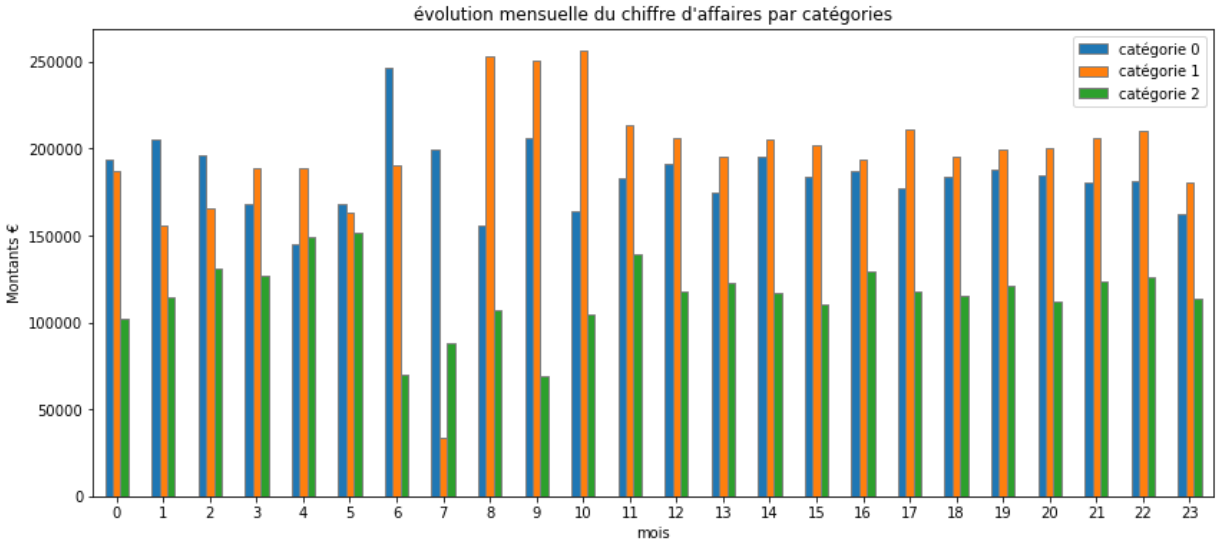
```
achats = achats.sort_values(["date"],ascending=True)
_2021 = achats.loc[achats['years']==2021,:].sort_values(by='date')
_2022 = achats.loc[achats['years']==2022,:].sort_values(by='date')
_2023 = achats.loc[achats['years']==2023,:].sort_values(by='date')

fig = plt.figure(figsize=(15,6))
plt.title("Evolution du chiffre d'affaire par an et par mois")
_2021.groupby(_2021['date'].dt.month).sum()['price'].plot(kind='line',color="black",
_2022.groupby(_2022['date'].dt.month).sum()['price'].plot(kind='line',color='red',ro
_2023.groupby(_2023['date'].dt.month).sum()['price'].plot(kind='line',color='green',
plt.legend(prop={'size':11})
fig.savefig("assets/graphiques/3_Evolution_du_CA_annee_mois.png")
```

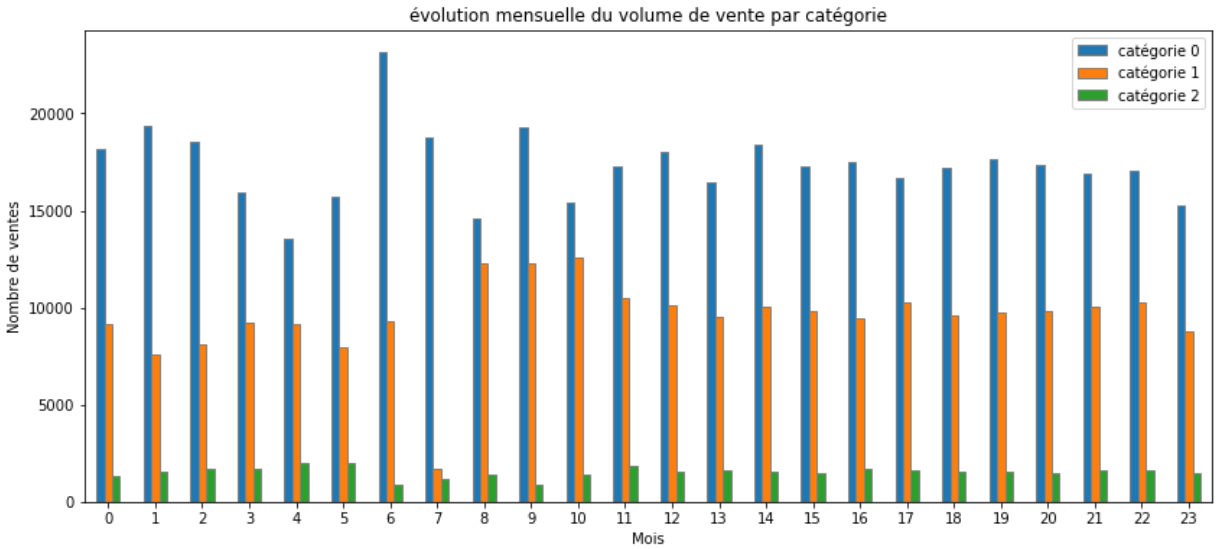


In [54]:
```
# Creation d'un dataframe specifique par categories avec fonction d'agregation:
agreg_mois = achats[["categ", "price", "years", "month"]]
agreg_mois = pd.pivot_table(agreg_mois, index=["years","month"], columns=["categ"],
agreg_mois.columns = ["years", "month", "categ_0", "categ_1", "categ_2", "vol_categ_
```

In [55]:
```
fig = plt.figure(figsize=(15,6))
# Graphique CA par mois:
agreg_mois[["categ_0", "categ_1", "categ_2"]].plot.bar(figsize=(14,6),edgecolor = "g
plt.legend(["catégorie 0", "catégorie 1", "catégorie 2"])
plt.xlabel("mois")
plt.ylabel("Montants €")
plt.title("évolution mensuelle du chiffre d'affaires par catégories")
plt.savefig("assets/graphiques/5.1 Evolution du CA par mois et par catégories.png")

# Graphique volume des ventes:
agreg_mois[["vol_categ_0", "vol_categ_1", "vol_categ_2"]].plot.bar(figsize=(14,6),ro
plt.legend(["catégorie 0", "catégorie 1", "catégorie 2"])
plt.xlabel("Mois")
plt.ylabel("Nombre de ventes")
plt.title("évolution mensuelle du volume de vente par catégorie")
plt.savefig("assets/graphiques/5.2 Evolution du volume des ventes par mois et catégo
```

```
<Figure size 1080x432 with 0 Axes>
```





In [56]:
```
# identifier les mois en fonction des données
# 7 = octobre = forte baisse
_2021 = achats.loc[achats['years']==2021,:].sort_values(by='date')
_2022 = achats.loc[achats['years']==2022,:].sort_values(by='date')
_2023 = achats.loc[achats['years']==2023,:].sort_values(by='date')

_2021['date'].dt.month.unique()
```

Out[56]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12], dtype=int64)
```

In [57]:
```
_2022['date'].dt.month.unique()
```

Out[57]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12], dtype=int64)
```

In [58]:
```
_2023['date'].dt.month.unique()
```

Out[58]:
```
array([1, 2], dtype=int64)
```

In [59]:
```
# Par catégories
agreg_mois = achats[["categ", "price", "years", "month","day"]]
agreg_mois = pd.pivot_table(agreg_mois, index=["years","month","day"], columns=["cat
agreg_mois.columns = ["years", "month","day", "categ_0", "categ_1", "categ_2", "vol_
```

In [60]:
```
# Nombre de ventes moyennes par jours et par mois:
print("Nombre de ventes moyennes par jour selon la catégories")
achats_jour = agreg_mois.groupby(["day","month","years"])["vol_categ_0", "vol_categ_
print(round(achats_jour,2))
```

```
Nombre de ventes moyennes par jour selon la catégories
vol_categ_0    569.42
vol_categ_1    311.19
vol_categ_2     49.98
dtype: float64
C:\Users\tomy\AppData\Local\Temp/ipykernel_10484/380238487.py:3: FutureWarning: Inde
xing with multiple keys (implicitly converted to a tuple of keys) will be deprecate
d, use a list instead.
  achats_jour = agreg_mois.groupby(["day","month","years"])["vol_categ_0", "vol_cate
g_1", "vol_categ_2"].sum().mean()
```

forte baisse en octobre de la catégorie 1 seulement

In [61]:
```
# Octobre 2021
octobre = achats.loc[(achats["years"] == 2021) & (achats["month"] == 10), ["day", "p
octobre = octobre.groupby(["day","categ"]).agg({"price":["sum","count"]}).reset_inde
octobre.columns = ["day", "categ", "ca", "volume"]
octobre = pd.pivot_table(octobre, columns=["categ"], index=["day"]).reset_index()
octobre.columns = ["day", 'ca_0', 'ca_1', 'ca_2', 'vol_0', 'vol_1', 'vol_2']
octobre
```

Out[61]:

	day	ca_0	ca_1	ca_2	vol_0	vol_1	vol_2
0	1	6950.50	7003.79	3104.05	663.0	344.0	38.0
1	2	7141.01	NaN	2041.12	661.0	NaN	28.0
2	3	6786.57	NaN	2206.48	648.0	NaN	31.0

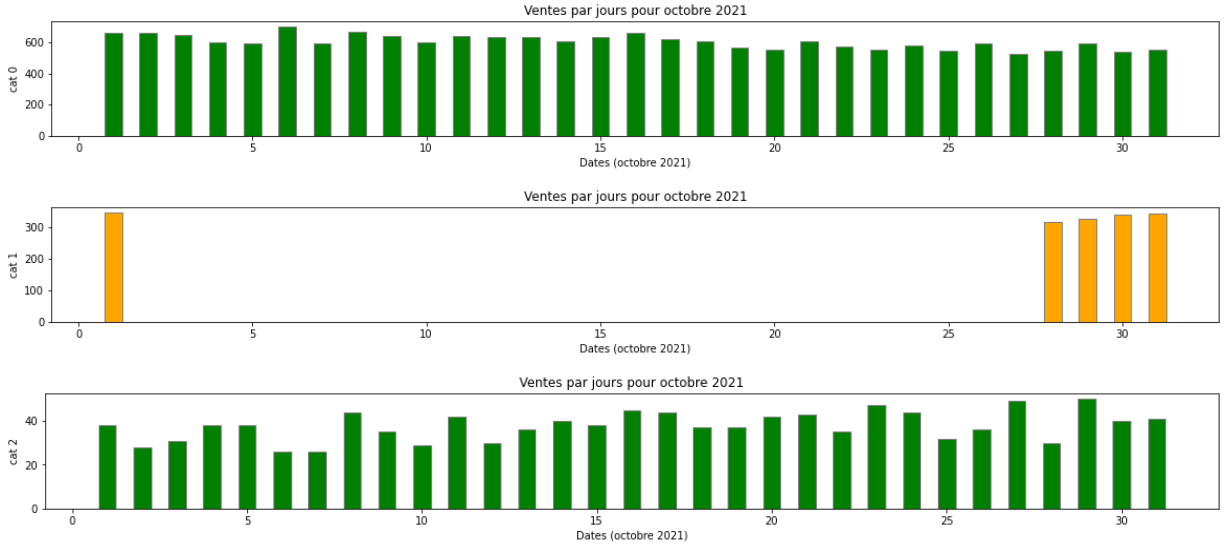
	day	ca_0	ca_1	ca_2	vol_0	vol_1	vol_2
3	4	6553.58	NaN	2746.08	603.0	NaN	38.0
4	5	6357.91	NaN	3032.55	594.0	NaN	38.0
5	6	7546.58	NaN	1944.11	702.0	NaN	26.0
6	7	6404.01	NaN	1787.07	597.0	NaN	26.0
7	8	7069.53	NaN	3137.82	669.0	NaN	44.0
8	9	6808.69	NaN	2616.67	640.0	NaN	35.0
9	10	6490.98	NaN	2334.67	600.0	NaN	29.0
10	11	7005.40	NaN	3225.16	642.0	NaN	42.0
11	12	6706.97	NaN	2264.18	633.0	NaN	30.0
12	13	6760.40	NaN	2666.82	633.0	NaN	36.0
13	14	6422.39	NaN	3047.39	606.0	NaN	40.0
14	15	6707.07	NaN	2701.75	634.0	NaN	38.0
15	16	6898.14	NaN	3330.88	661.0	NaN	45.0
16	17	6492.60	NaN	3065.38	625.0	NaN	44.0
17	18	6376.43	NaN	2707.29	608.0	NaN	37.0
18	19	5805.69	NaN	2816.00	567.0	NaN	37.0
19	20	5912.76	NaN	3118.42	555.0	NaN	42.0
20	21	6438.85	NaN	3132.94	610.0	NaN	43.0
21	22	6051.86	NaN	2416.00	572.0	NaN	35.0
22	23	5704.26	NaN	3206.38	555.0	NaN	47.0
23	24	6036.65	NaN	3886.54	584.0	NaN	44.0
24	25	5817.82	NaN	2258.22	545.0	NaN	32.0
25	26	6345.00	NaN	2685.78	592.0	NaN	36.0
26	27	5822.24	NaN	3967.29	530.0	NaN	49.0
27	28	5837.61	6317.99	2602.82	548.0	316.0	30.0
28	29	6410.57	6425.18	3617.88	595.0	326.0	50.0
29	30	5734.62	6753.69	2898.47	542.0	338.0	40.0
30	31	5924.07	7261.67	3219.38	555.0	342.0	41.0

Erreur de collecte de données possible : pas de vente de produits de catégorie 1 du 2 au 27 octobre

In [62]:

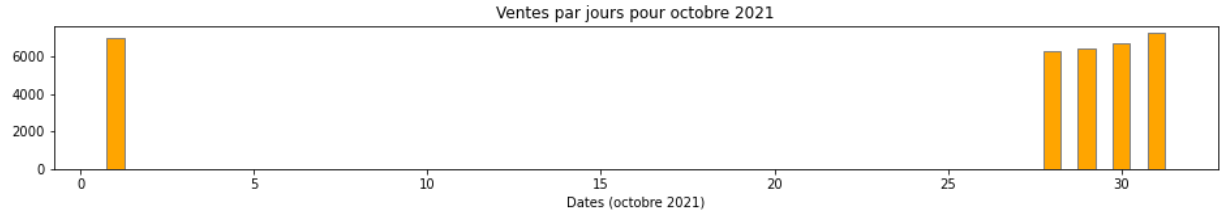
```
colors = ["green",'orange','green']
for i, cat in enumerate([0,1,2]):
    plt.figure(figsize=(44,2))
    plt.subplot(1,2,2)
    plt.bar(octobre["day"], octobre[f'vol_{cat}'], width=0.5,edgecolor = "grey",colo
```

```
    plt.xlabel("Dates (octobre 2021)")
    plt.ylabel(f'cat {cat}')
    plt.title("Ventes par jours pour octobre 2021")
    plt.show()
```



On remarque que les ventes de livres de catégorie 1 sont inexistantes au mois d'octobre 2021, cette baisse ne se retrouve pas plus tard, il semble qu'il y ait eu un problème.

In [63]:
```
plt.figure(figsize=(16,2))
plt.bar(octobre["day"], octobre['ca_1'], width=0.5,edgecolor = "grey",color="orange"
plt.xlabel("Dates (octobre 2021)")
plt.title("Ventes par jours pour octobre 2021")
plt.savefig("assets/graphiques/6.1 octobre données manquantes pour la categorie 1.pn
plt.show()
```



Décomposition en moyenne mobile pour évaluer la tendance globale

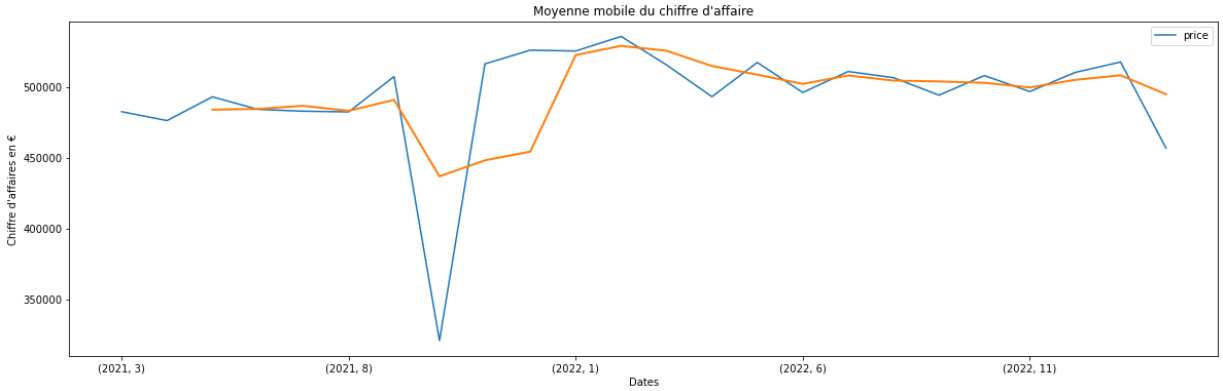
Décomposition en moyenne mobile

La moyenne mobile, ou moyenne glissante, est un type de moyenne statistique utilisée pour analyser des séries ordonnées de données, le plus souvent des séries temporelles, en supprimant les fluctuations transitoires de façon à en souligner les tendances à plus long terme. Cette moyenne est dite mobile parce qu'elle est recalculée de façon continue, en utilisant à chaque calcul un sous-ensemble d'éléments dans lequel un nouvel élément remplace le plus ancien ou s'ajoute au sous-ensemble.

Ce type de moyenne est utilisé généralement comme méthode de lissage de valeurs.

In [64]:
```
# Decomposition en moyenne mobile:
moyenne_mobile = ca_par_mois.rolling(3).mean()
print(round(moyenne_mobile,2))

#Graphique par mois
plt.figure(figsize = (20,6))
ca_par_mois.plot()
plt.title("Evolution mensuelle du chiffre d'affaires")
plt.legend()

moyenne_mobile.plot(linewidth=2)
plt.xlabel("Dates")
plt.ylabel("Chiffre d'affaires en €")
plt.title("Moyenne mobile du chiffre d'affaire")

plt.savefig("assets/graphiques/7 decomposition en moyenne mobile")
```

```
years  month
2021   3              NaN
       4              NaN
       5        483934.36
       6        484477.01
       7        486685.75
       8        483136.18
       9        490870.21
       10       436867.98
       11       448165.62
       12       454374.50
2022   1        522547.93
       2        529019.18
       3        525545.58
       4        514795.54
       5        508665.89
       6        502172.43
       7        508093.83
       8        504512.08
       9        503884.88
       10       502923.10
       11       499665.65
       12       505023.98
2023   1        508221.58
       2        494879.87
Name: price, dtype: float64
```



In [65]:
```
achats
```

Out[65]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	years	mont
468405	0_1259	2021-03-01 00:01:07.843138	s_1	c_329	11.99	0.0	f	1967.0	55.0	2021	

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	years	mont
497884	0_1390	2021-03-01 00:02:26.047414	s_2	c_664	19.37	0.0	m	1960.0	62.0	2021	
221213	0_1352	2021-03-01 00:02:38.311413	s_3	c_580	4.50	0.0	m	1988.0	34.0	2021	
316014	0_1458	2021-03-01 00:04:54.559692	s_4	c_7912	6.55	0.0	f	1989.0	33.0	2021	
595898	0_1358	2021-03-01 00:05:18.801198	s_5	c_2033	16.49	0.0	f	1956.0	66.0	2021	
...	...	...	...	...	...	...	...	...	...	...	
163305	1_508	2023-02-28 23:49:03.148402	s_348444	c_3573	21.92	1.0	f	1996.0	26.0	2023	
153419	2_37	2023-02-28 23:51:29.318531	s_348445	c_50	48.99	2.0	f	1994.0	28.0	2023	
476829	1_695	2023-02-28 23:53:18.929676	s_348446	c_488	26.99	1.0	f	1985.0	37.0	2023	
428188	0_1547	2023-02-28 23:58:00.107815	s_348447	c_4848	8.99	0.0	m	1953.0	69.0	2023	
398652	0_1398	2023-02-28 23:58:30.792755	s_348435	c_3575	4.52	0.0	f	1981.0	41.0	2023	

679332 rows × 12 columns

In [66]:
```
# saisonnalité et tendance
achats.reset_index(inplace=True)
achats['dates'] = pd.to_datetime(achats["date"])
achats = achats.set_index("date")
achats
```

Out[66]:

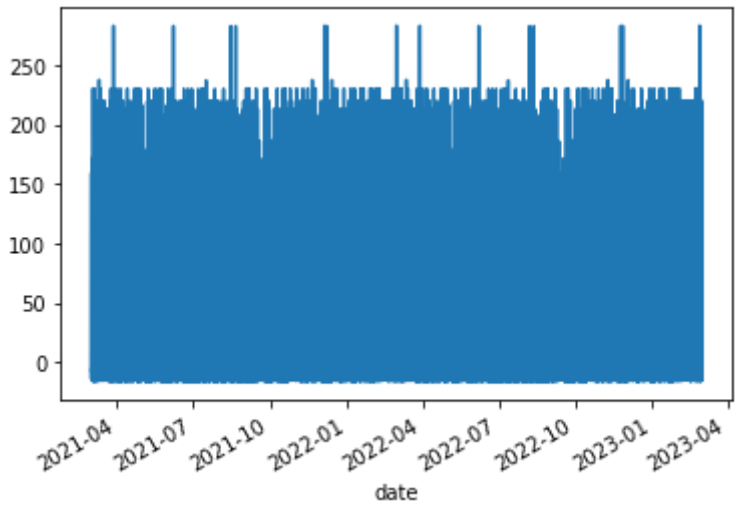
date	index	id_prod	session_id	client_id	price	categ	sex	birth	age	years	mon
2021-03-01 00:01:07.843138	468405	0_1259	s_1	c_329	11.99	0.0	f	1967.0	55.0	2021	
2021-03-01 00:02:26.047414	497884	0_1390	s_2	c_664	19.37	0.0	m	1960.0	62.0	2021	
2021-03-01 00:02:38.311413	221213	0_1352	s_3	c_580	4.50	0.0	m	1988.0	34.0	2021	
2021-03-01 00:04:54.559692	316014	0_1458	s_4	c_7912	6.55	0.0	f	1989.0	33.0	2021	
2021-03-01 00:05:18.801198	595898	0_1358	s_5	c_2033	16.49	0.0	f	1956.0	66.0	2021	
...	...	...	...	...	...	...	...	...	...	...	
2023-02-28 23:49:03.148402	163305	1_508	s_348444	c_3573	21.92	1.0	f	1996.0	26.0	2023	
2023-02-28 23:51:29.318531	153419	2_37	s_348445	c_50	48.99	2.0	f	1994.0	28.0	2023	

date	index	id_prod	session_id	client_id	price	categ	sex	birth	age	years	mon
2023-02-28 23:53:18.929676	476829	1_695	s_348446	c_488	26.99	1.0	f	1985.0	37.0	2023	
2023-02-28 23:58:00.107815	428188	0_1547	s_348447	c_4848	8.99	0.0	m	1953.0	69.0	2023	
2023-02-28 23:58:30.792755	398652	0_1398	s_348435	c_3575	4.52	0.0	f	1981.0	41.0	2023	

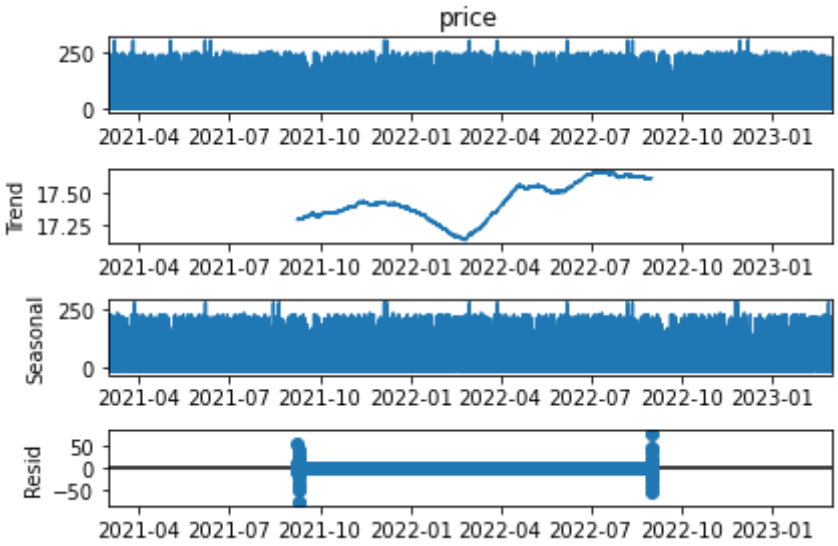
679332 rows × 13 columns

In [67]:
```
# Saisonnalité des prix
import statsmodels.api as sm
from matplotlib import rcParams
decomposition = sm.tsa.seasonal_decompose(achats["price"],period = 339600)

decomposition.seasonal.plot()
```

Out[67]:
```
<AxesSubplot:xlabel='date'>
```



In [68]:
```
sm.tsa.seasonal_decompose(achats["price"],period=339600).plot()
plt.savefig("assets/graphiques/8.2 saisonnalité.png")
```



Zoom sur les références

In [69]:
```
reference_achats = achats.groupby(["id_prod"])["session_id"].count()
reference_achats.sort_values()
```

Out[69]:
```
id_prod
0_549        1
0_2201       1
2_23         1
0_1284       1
0_1683       1
           ...
1_425     2096
1_498     2128
1_414     2180
1_417     2189
1_369     2252
Name: session_id, Length: 3266, dtype: int64
```

Les tops

In [70]:
```
# Les 10 produits les plus vendus
print("la liste des 10 produits les plus vendus : \n",reference_achats.nlargest(10))
```

```
la liste des 10 produits les plus vendus :
 id_prod
1_369    2252
1_417    2189
1_414    2180
1_498    2128
1_425    2096
1_403    1960
1_412    1951
1_413    1945
1_406    1939
1_407    1935
Name: session_id, dtype: int64
```

In [71]:
```
# Les 10 produits les moins vendus
print("la liste des 10 produits les mois vendus : \n",reference_achats.nsmallest(10)
```

```
la liste des 10 produits les mois vendus :
 id_prod
0_1151    1
0_1284    1
0_1379    1
0_1498    1
0_1539    1
0_1601    1
0_1633    1
0_1683    1
0_1728    1
0_2201    1
Name: session_id, dtype: int64
```

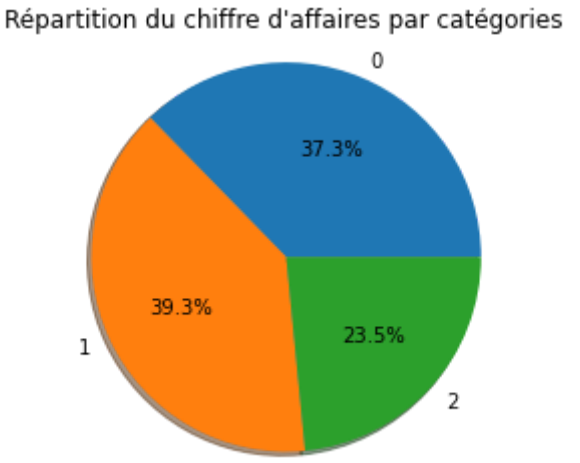
- **Les plus vendus appartiennent à la catégorie 1**
- **Les moins vendus appartiennent à la catégorie 0**

La répartition par catégorie

In [72]:
```
# Repartition des references parmis les categories:
repartition_categ = products.groupby(["categ"])["id_prod"].count()
print("Répartitions des références par catégories:\n",repartition_categ)

# Volume de ventes par categories:
prix_categ = products.groupby(["categ"])["price"].mean()
print("Prix moyen d'un livre selon sa catégorie:\n",prix_categ)

# Prix moyen par categories:
prix_categ = products.groupby(["categ"])["price"].mean()
print("Prix moyen d'un livre selon sa catégorie:\n",prix_categ)

# CA par categories:
ca_categ = achats.groupby(achats["categ"])["price"].sum()
print("Chiffre d'affaires par catégorie\n",ca_categ)
```

```
Répartitions des références par catégories:
 categ
0    2308
1     739
2     239
Name: id_prod, dtype: int64
Prix moyen d'un livre selon sa catégorie:
 categ
0     11.732795
1     25.531421
2    108.354686
Name: price, dtype: float64
Prix moyen d'un livre selon sa catégorie:
 categ
0     11.732795
1     25.531421
2    108.354686
Name: price, dtype: float64
Chiffre d'affaires par catégorie
 categ
0.0    4421938.76
1.0    4653722.69
2.0    2780275.02
Name: price, dtype: float64
```

In [73]:
```
fig1, ax1 = plt.subplots()
ax1.pie(ca_categ,
        labels=["0","1","2"],
        autopct="%1.1f%%",
        shadow=True)
ax1.axis("equal")
plt.title("Répartition du chiffre d'affaires par catégories")
plt.savefig("assets/graphiques/9 Répartition du CA par catégories.png")
plt.show()
```



- Le prix moyens des livres par catégories :
  - 0 11.732795
  - 1 25.531421
  - 2 108.354686
- Répartition par nb références
  - 0 2308
  - 1 739
  - 2 239
- Apport au chiffre d'affaire par catégorie
  - 0 4419730.97
  - 1 4653722.69
  - 2 2780275.02

CCL :

- par catégories :
  - catégorie 0 :
    - dispose du plus grand nombre de référence : 2308 références
    - le prix moyen est le plus faible 11.73
    - apporte un grande partie du ca : 4 419 970,97
  - catégorie 1 :
    - dispose un nombre important de références mais nettement moindre que la catégorie 0
    - le prix moyen est plus important : 25.53
    - apporte une grande partie du ca : 4 653 722,69
  - catégorie 2 :
    - dispose du nombre le plus faible de références : 239
    - le prix moyen est le plus élevé : 108.35
    - apporte au ca la plus faible part: 2 780 275.02

Les profils de nos clients

Répartition du chiffre d'affaire entre les clients : Courbe de Lorenz et coefficient de Gini :

reference a voir

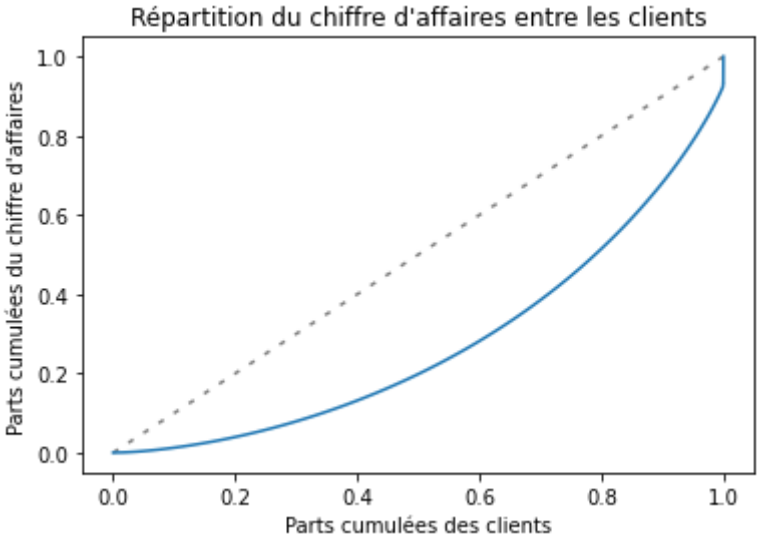
In [74]:
```
# Je veux savoir combien chaque client a depensé triée par ordre croissant
achatsClients = df.groupby('client_id')['price'].sum().sort_values().reset_index()
achatsClients
```

Out[74]:

	client_id	price
0	c_587	0.00
1	c_4447	0.00
2	c_7584	0.00
3	c_4358	0.00
4	c_6735	0.00
...	...	...
8616	c_3263	5276.87
8617	c_3454	113667.90
8618	c_6714	153658.86
8619	c_4958	289760.34
8620	c_1609	324033.35

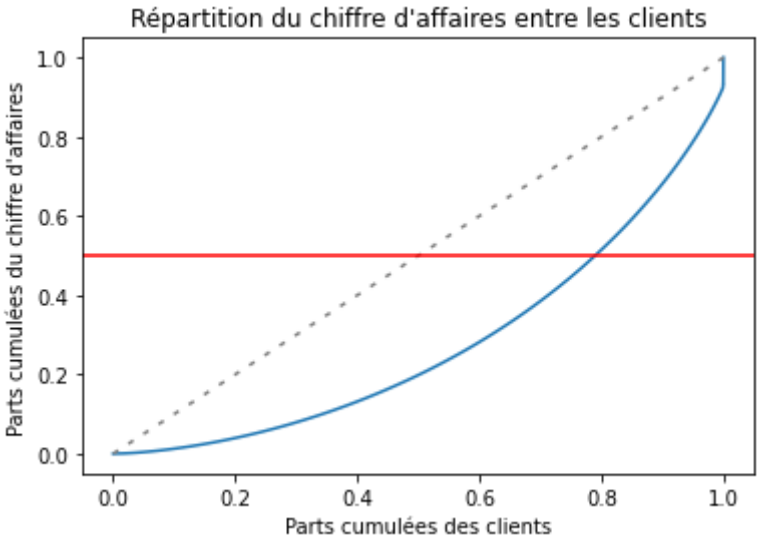
8621 rows × 2 columns

In [75]:
```
# Répartition du chiffre d'affaires entre les clients
achatsClients = df.groupby('client_id')['price'].sum().sort_values()
n = len(achatsClients)
lorenz = np.cumsum(achatsClients)/ achatsClients.sum()
lorenz = np.append([0],lorenz) # La courbe de Lorenz commence à 0

plt.plot(np.linspace(0,1,len(lorenz)), np.linspace(0,1,len(lorenz)), color ="grey",

xaxis = np.linspace(0-1/n,1+1/n,n+1) #Il y a un segment de taille n pour chaque indi
plt.plot(xaxis,lorenz,drawstyle='steps-post')
plt.ylabel("Parts cumulées du chiffre d'affaires")
plt.xlabel("Parts cumulées des clients")
plt.title("Répartition du chiffre d'affaires entre les clients")
plt.savefig("assets/graphiques/10 Courbe de Lorenz de la répartition des achats clie
plt.show()
```



In [76]:

```
#Je veux tracer la médiale : la valeur de la dépance médiale
plt.plot(xaxis,lorenz,drawstyle='steps-post')
plt.title("Répartition du chiffre d'affaires entre les clients")
plt.ylabel("Parts cumulées du chiffre d'affaires")
plt.xlabel("Parts cumulées des clients")
plt.plot(np.linspace(0,1,len(lorenz)), np.linspace(0,1,len(lorenz)),color ="grey", d
plt.axhline(0.5, color='red')
plt.savefig("assets/graphiques/10 Courbe de Lorenz Médiale et 80-20.png")
plt.show()
print("La loi de Pareto des 80/20 ou 20/80 s'applique dans ce cas")
print("Maintenant on sait que la somme des achats des acheteurs inférieurs à l'achet
```




```
La loi de Pareto des 80/20 ou 20/80 s'applique dans ce cas
Maintenant on sait que la somme des achats des acheteurs inférieurs à l'acheteur méd
ial vaut 50 % de la somme de tous les acheteurs, et évidemment, la somme des achats
des acheteurs supérieurs à l'acheteur médial vaut 50 % de la somme de tous les achet
eurs.
CONCLUSION : Il y a un petit nombre de gros acheteurs
20% des acheteurs font 50% du chiffre d'affaire
 et 80% font 50% du CA
```

Indice de gini

In [77]:

```
AUC = (lorenz.sum() -lorenz[-1]/2 -lorenz[0]/2)/n # Surface sous la courbe de Lorenz
S = 0.5 - AUC # surface entre la première bissectrice et le courbe de Lorenz
```

```
gini = 2*S
print("L'indice de Gini : ",gini,"montre la répartition des acheteurs est hétéroclit
```

```
L'indice de Gini :  0.44774343738574474 montre la répartition des acheteurs est hété
roclite : qui peut s'expliquer par différentes hypothèses, les types d'acheteurs (oc
casionnels, professionnels, passionées... dans tous les cas il s'éloigne fortement d
e 0 donc le poids des produits dans le chiffre d'affaire est très inégal
```

In [78]:
```
meilleurs_clients = achats.groupby(["client_id"])["price"].sum()
```

L'indice de Gini : 0.4477350952112892

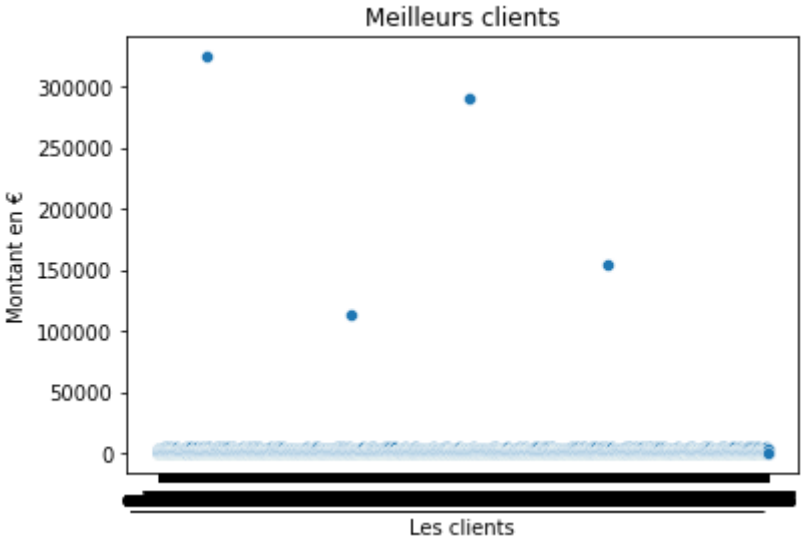
- la répartition des acheteurs est hétéroclite : qui peut s'expliquer par différentes hypothèses, les types d'acheteurs (occasionnels, professionnels (libraires, bibliotécaires, revendeurs...), passionées...
- dans tous les cas il s'éloigne fortement de 0 donc le poids des produits dans le chiffre d'affaire est très inégal

In [79]:
```
# meilleurs clients
meilleurs_clients = achats.groupby(["client_id"])["price"].sum()
meilleurs_clients.nlargest(10)
```

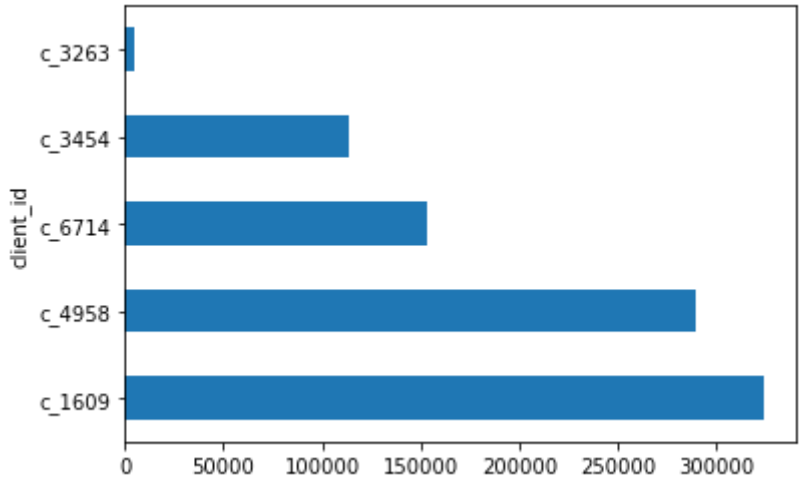
Out[79]:
```
client_id
c_1609    324033.35
c_4958    289760.34
c_6714    153658.86
c_3454    113667.90
c_3263      5276.87
c_1570      5271.62
c_2899      5214.05
c_2140      5208.82
c_7319      5155.77
c_8026      5092.57
Name: price, dtype: float64
```

In [80]:
```
# Graphique:
sns.scatterplot(data= meilleurs_clients)
plt.ylabel("Montant en €")
plt.xlabel("Les clients")
plt.title("Meilleurs clients")
```

Out[80]:
```
Text(0.5, 1.0, 'Meilleurs clients')
```



In [81]:
```
# Mise en avant des 4 meilleurs clients
meilleurs_clients.nlargest(5).plot(kind="barh")
plt.savefig("assets/graphiques/16 meilleurs clients")
```



- Le client avec l'id : c_1609 est un homme né en 1980 agé de 42 ans qui a passé 25488 commandes et a dépensé au total 324033.35 euros. Il peut s'agir d'un revendeur, un libraire ...
- Lui comme les 4 meilleurs clients ont dépenser un montant significatif sur 3 ans qui permet de voir dans ces acheteurs des clients particulier et non des particuliers:
  - c_1609 324033.35 euros
  - c_3454 113637.93 euros
  - c_4958 289760.34 euros
  - c_6714 153598.92 euros

In [82]:
```
# Apport des gros clients au Chiffre d'affaire
print("Les meilleurs clients représentent :",round(meilleurs_clients.nlargest(4).sum
```

```
Les meilleurs clients représentent : 7.43 % du Chiffre d'affaire total
```

In [83]:
```
# nombre d'achats par client
nb_achat_moyen = achats.groupby(["client_id"])["session_id"].count().mean()
print("Le nombre moyen d'achats par client : ",nb_achat_moyen)
```

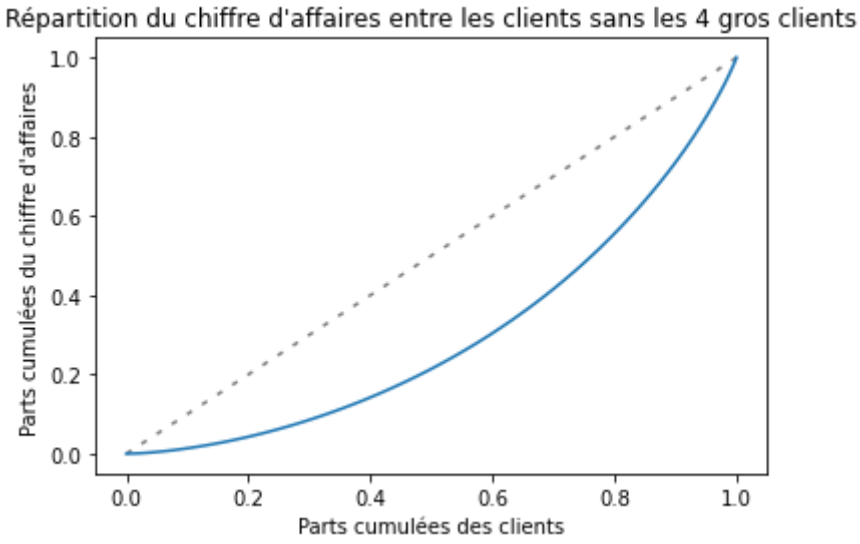
```
Le nombre moyen d'achats par client :  78.99209302325582
```

In [84]:
```
# montant moyen par client
montant_achats_moyen = achats.groupby(['client_id'])["price"].sum().mean()
print("Le montant moyen par client : ",montant_achats_moyen)
```

```
Le montant moyen par client :  1378.597263953488
```

In [85]:
```
# DF sans les 4 gros clients
df_sgc= df
df_sgc.drop(df_sgc.loc[df_sgc['client_id']=='c_1609'].index, inplace=True)
df_sgc.drop(df_sgc.loc[df_sgc['client_id']=='c_3454'].index, inplace=True)
df_sgc.drop(df_sgc.loc[df_sgc['client_id']=='c_4958'].index, inplace=True)
df_sgc.drop(df_sgc.loc[df_sgc['client_id']=='c_6714'].index, inplace=True)
```

In [86]:
```
# Répartition du CA entre les clients sans les 4 gros clients (df_sgc)
achatsClients = df_sgc.groupby('client_id')['price'].sum().sort_values()
n = len(achatsClients)
lorenz = np.cumsum(achatsClients)/ achatsClients.sum()
lorenz = np.append([0],lorenz) # La courbe de Lorenz commence à 0

plt.plot(np.linspace(0,1,len(lorenz)), np.linspace(0,1,len(lorenz)), color ="grey",

xaxis = np.linspace(0-1/n,1+1/n,n+1) #Il y a un segment de taille n pour chaque indi
plt.plot(xaxis,lorenz,drawstyle='steps-post')
plt.ylabel("Parts cumulées du chiffre d'affaires")
plt.xlabel("Parts cumulées des clients")
plt.title("Répartition du chiffre d'affaires entre les clients sans les 4 gros clien
plt.savefig("assets/graphiques/10.2 Courbe de Lorenz de la répartition des achats cl
plt.show()
```



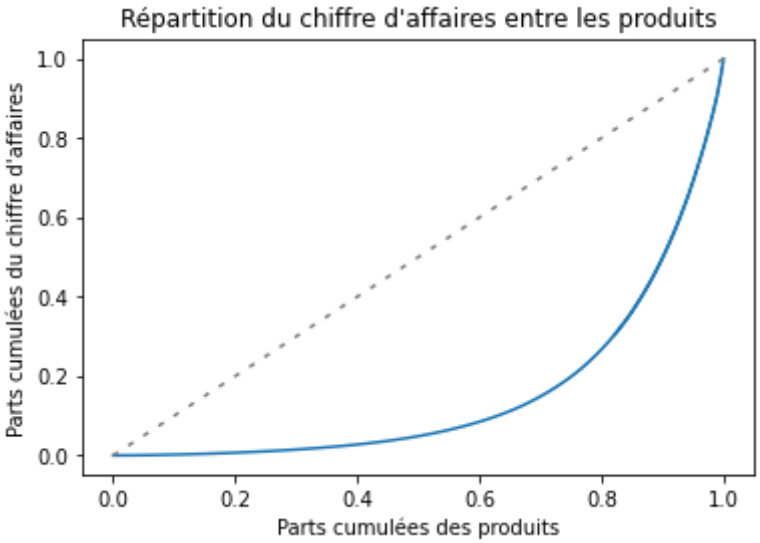
In [87]:
```
AUC = (lorenz.sum() -lorenz[-1]/2 -lorenz[0]/2)/n # Surface sous la courbe de Lorenz
S = 0.5 - AUC # surface entre la première bissectrice et le courbe de Lorenz
gini = 2*S
print("L'indice de Gini : ",gini)
```

```
L'indice de Gini :  0.4040857644394076
```

Répartition du Chiffre d'affaire entre les produits : Courbe de Lorenz

In [88]:
```
# Courbe de Lorentz: (Repartition du CA par produits)
nbr_ventes_ref = achats.groupby(["id_prod"])["session_id"].count()
lorenz_CA_prod = np.cumsum(np.sort(nbr_ventes_ref)) / nbr_ventes_ref.sum()
lorenz_CA_prod = np.append([0], lorenz_CA_prod)
plt.plot(np.linspace(0,1,len(lorenz_CA_prod)), lorenz_CA_prod, drawstyle='steps-post
plt.plot(np.linspace(0,1,len(lorenz_CA_prod)), np.linspace(0,1,len(lorenz_CA_prod)),
plt.ylabel("Parts cumulées du chiffre d'affaires")
plt.xlabel("Parts cumulées des produits")
plt.title("Répartition du chiffre d'affaires entre les produits")
plt.savefig("assets/graphiques/10 Courbe de Lorenz produits.png")

aire_ss_courbe_CA_prod = lorenz_CA_prod[:-1].sum() / len(lorenz_CA_prod)
S_ca_prod = 0.5 - aire_ss_courbe_CA_prod
gini_CA_prod = 2* S_ca_prod
print("indice de Gini:",round(gini_CA_prod,3))
```

```
indice de Gini: 0.694
```

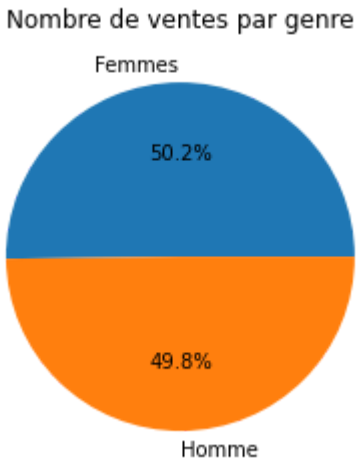


In [89]:
```
print("L'indice de Gini est loin de 0 le poids des produits dans le chiffre d'affair
print("Loi des 80/20 : 20 % des produits fournissent 80% du Chiffre d'affaires")
```

```
L'indice de Gini est loin de 0 le poids des produits dans le chiffre d'affaires tota
l est donc très inégal.
Loi des 80/20 : 20 % des produits fournissent 80% du Chiffre d'affaires
```

Répartition par sexe des clients

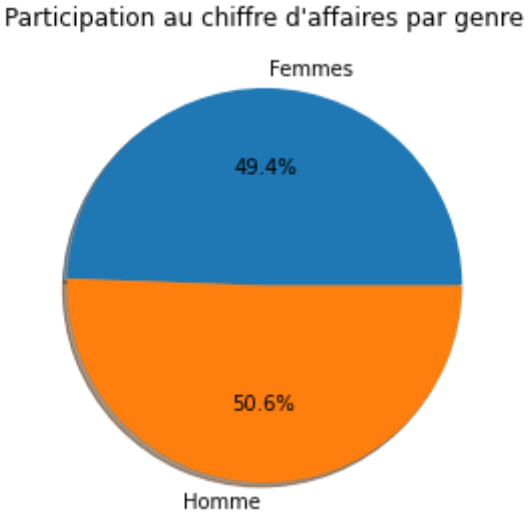
In [130…
```
#Répartition du nombre d'achats en fonction du genre du client
achats['sex'].value_counts(normalize=False).plot(kind='pie',rot='0',
                                                 labels=["Femmes","Homme"],
                                                 autopct="%1.1f%%")
plt.xlabel("")
plt.ylabel("")
plt.title("Nombre de ventes par genre")
plt.savefig("assets/graphiques/9.2 Nombre de vente par genre.png")
plt.show()
```



In [91]:

```
repartition_achats_sexe = achats.groupby(["sex"])["price"].sum()
plt.figure()
fig, ax = plt.subplots()
ax.pie(repartition_achats_sexe,
        labels=["Femmes","Homme"],
        autopct="%1.1f%%",
        shadow=True)
ax.axis("equal")
plt.title("Participation au chiffre d'affaires par genre\n")
plt.savefig("assets/graphiques/ 12 Partitcipation au CA par genre.png")
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [92]:

```
# Dépense par genres:
repartition_achats_genre = achats.groupby(['sex'])['price'].sum()
print(f"Montant total des achats par genre en euros:\n{repartition_achats_genre}")

print("_____________________________________________________")
# Nombre d'achats par genre
nb_achats_genre = achats.groupby(["sex"])["price"].count()
print(f"Nombre total d'achats par genre\n {nb_achats_genre}")
print("_____________________________________________________")

# moyenne des dépenses par genre
mean_achats_genre = achats.groupby(["sex"])["price"].mean()
print(f"Moyenne des achats par genre\n {mean_achats_genre}")
```

```
Montant total des achats par genre en euros:
```

```
sex
f    5860851.96
m    5995084.51
Name: price, dtype: float64
```

```
Nombre total d'achats par genre
 sex
f    338402
m    340930
Name: price, dtype: int64
```

```
Moyenne des achats par genre
 sex
f    17.319200
m    17.584503
Name: price, dtype: float64
```

le lien entre le genre d'un client et les catégories des livres achetés

CHI2 méthode 1 et 2 pour verifier

In [93]:

```
# Table de contingence pour Chi²:
# chi2 table de contingence
X = "sex"    # qualitative
Y = "categ" # qualitative

cont = achats[[X,Y]].pivot_table(index=X,columns=Y,aggfunc=len,margins=True,margins_
cont
```

Out[93]:

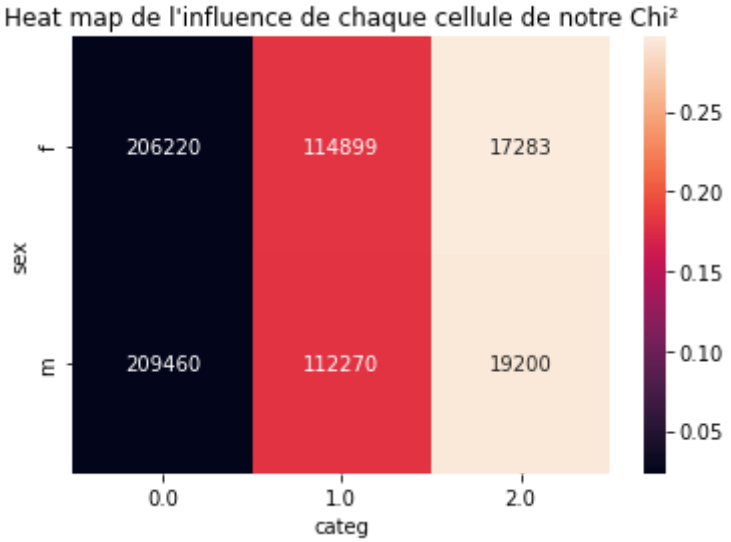
categ	0.0	1.0	2.0	Total
sex				
f	206220	114899	17283	338402
m	209460	112270	19200	340930
Total	415680	227169	36483	679332

In [94]:

```
# Chi²:
import scipy.stats as st
st_chi2, st_p, st_dof, st_exp = st.chi2_contingency(cont)

print("Les fréquences théoriques sont:\n", st_exp)
# degré de liberté ddl = nbligne-1 * nbColonne-1 ==> 4-1*3-1 = 3*2 = 6 = ddl = st_do
print(f"Ce qui donne un X² de {st_chi2:.3f} à", st_dof, "ddl")
print(f"soit une p-value de", st_p)
```

```
Les fréquences théoriques sont:
 [[207066.56444861 113161.81769444  18173.61785695 338402.        ]
 [208613.43555139 114007.18230556  18309.38214305 340930.        ]
 [415680.         227169.          36483.         679332.        ]]
Ce qui donne un X² de 147.003 à 6 ddl
soit une p-value de 3.327978654785752e-29
```

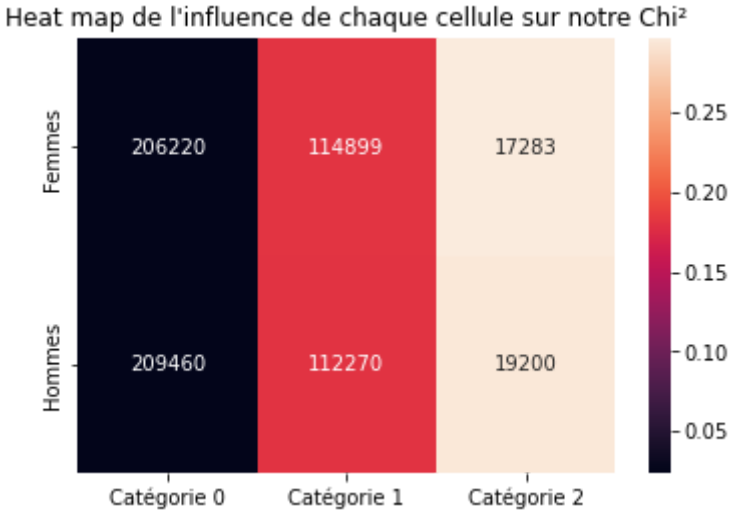
In [95]:
```
tx = cont.loc[:,["Total"]]
ty = cont.loc[["Total"],:]
n = len(achats)
indep = tx.dot(ty) / n

c = cont.fillna(0) # On remplace les valeurs nulles par 0
measure = (c-indep)**2/indep
xi_n = measure.sum().sum()
table = measure/xi_n
sns.heatmap(table.iloc[:-1,:-1],annot=c.iloc[:-1,:-1],fmt='d')
plt.title("Heat map de l'influence de chaque cellule de notre Chi²")
plt.savefig("assets/graphiques/ 12 lien entre genre et catégorie de livre acheté")
plt.show()
```



In [96]:
```
# khi2 entre sex et montant p4C5
X = "sex"
Y = "categ"
cont = achats[[X,Y]].pivot_table(index=X,columns=Y,aggfunc=len,margins=True,margins_
```

In [97]:
```
# On crée categ_sex pour enlever les totaux et pouvoir fair un heatmap
categ_sex=achats[[X,Y]].pivot_table(index=X,columns=Y,aggfunc=len,margins=False)
# Heat map pour identifier les cellules les plus influantes sur le Chi²:
# Création et initialisation du tableau a 0
len_i = categ_sex.shape[0] # 2
len_j = categ_sex.shape[1] # 3
xij = np.zeros((len_i, len_j)) # initialisation à 0
#xij
# boucle
for i in range(len_i):
    for j in range(len_j):
        xij[i,j] = (categ_sex.values[i,j]-st_exp[i,j])**2 / st_exp[i,j]
#xij
# heatmap
sns.heatmap(xij/st_chi2,
            annot=categ_sex, # on ecrit dans les cases
            fmt='d', # on enlève l'écriture scientifique (format decimal)
            yticklabels=["Femmes","Hommes"],# label des x
            xticklabels=["Catégorie 0", "Catégorie 1", "Catégorie 2"]) # label des y
plt.title("Heat map de l'influence de chaque cellule sur notre Chi²") #titre
plt.plot()
```

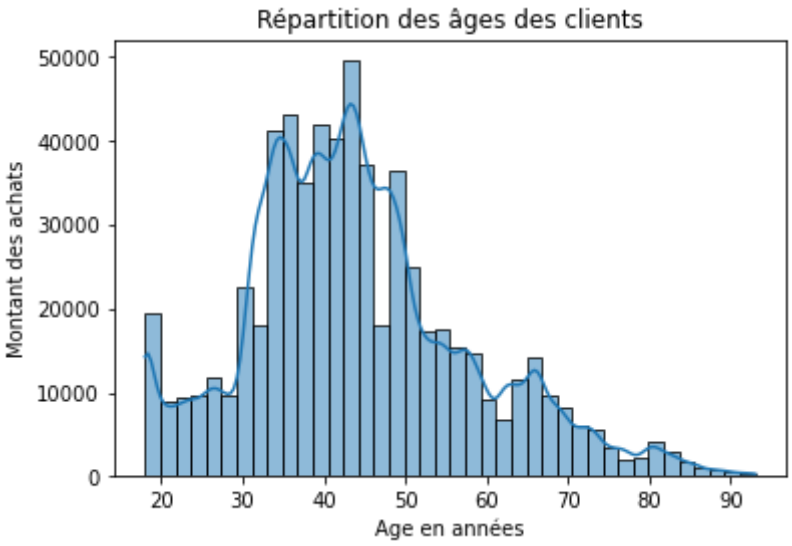
Out[97]:
```
[]
```



- **L'hypothèse nulle : Il n'existe pas de lien entre les catégories achetées et le genre.**
  - Que l'on prenne un seuil à 5% (0,05) ou 1% (0,01), la p-value sera ici toujours très inférieure au seuil de significativité,
  - **l'hypothèse nulle est rejetée**,
- **L'hypothèse alternative HA est retenue les catégories des livres achetées dépendent du genre de l'acheteur.** Il y a bien une dépendance entre le genre et le sexe (cf catégorie 2)

le lien entre l'âge des clients et le montant total des achats

In [98]:
```
sns.histplot(data=df["age"],
             bins=40,
             kde=True)
plt.title("Répartition des âges des clients")
plt.ylabel("Montant des achats")
plt.xlabel("Age en années")
plt.savefig("assets/graphiques/13.1 Lien entre l'age des clients et le montant des a
plt.show()
```



In [99]:
```
# test de student
```

In [132…
```
from scipy.stats import ttest_ind
ttest_ind(achats['age'], achats['price'])
```

Out[132…
```
Ttest_indResult(statistic=966.2513360457239, pvalue=0.0)
```

In [100…
```
# Ajout des tranches d'ages au df achats:
achats["age_rank"] = pd.cut(x= achats["age"], bins=[17,37,57,77,100], include_lowest
#achats.head()

# Repartition des montants des achats d'ages et montant:
repartition_achats_age_rank = achats.groupby(["age_rank"])["price"].sum()
print("montant total des achats par tranches d'age:\n", repartition_achats_age_rank)

# Nombre d'achats par tranches d'ages:
nbr_achats_age_rank = achats.groupby(["age_rank"])["price"].count()
print("nombre total d'achats par tranches d'age:\n",nbr_achats_age_rank)

# Montant moyen des achats par tranches d'ages:
moy_achats_age_rank = achats.groupby(["age_rank"])["price"].mean()
print("montant moyen des achats par tranches d'age:\n", moy_achats_age_rank)
```

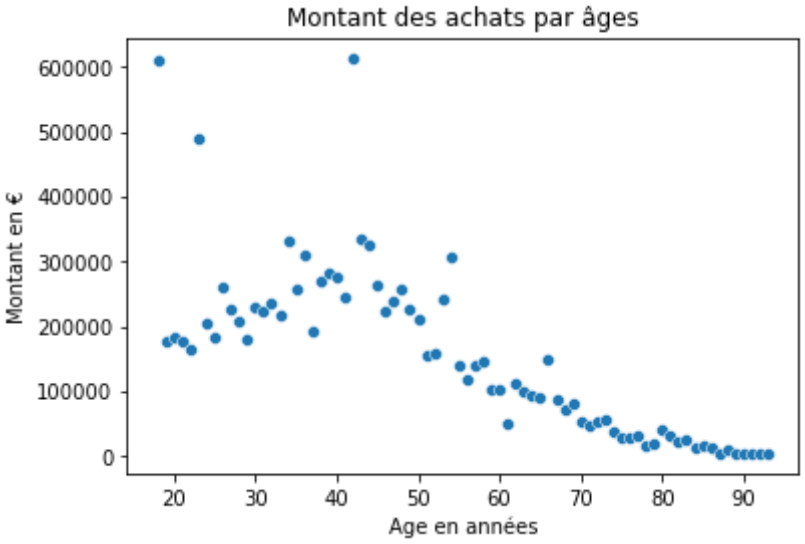
```
montant total des achats par tranches d'age:
 age_rank
(17, 37]     5064750.89
(37, 57]     5031521.58
(57, 77]     1526947.87
(77, 100]     232716.13
Name: price, dtype: float64
nombre total d'achats par tranches d'age:
 age_rank
(17, 37]     213655
(37, 57]     360725
(57, 77]      91110
(77, 100]     13842
Name: price, dtype: int64
montant moyen des achats par tranches d'age:
 age_rank
(17, 37]     23.705277
(37, 57]     13.948358
(57, 77]     16.759388
(77, 100]    16.812320
Name: price, dtype: float64
```

- Les 17-37 ans ont les panier moyens les plus élevés (23.7)
- Les 37-57 ans ont les paniers moyens les moins élevés (13.9)
- Ces 2 tranches rapportent autant de CA l'une que l'autres (17, 37] Les 37-57 achète plus de produits que les 17-37

In [101…
```
# Lien entre l'age et le montant des achats:
age = achats.groupby(["age"])["client_id"].count()

# total des achats par age
tot_achats_age = achats.groupby(["age"])["price"].sum()

# Correlation entre l'age et le montant total des achats:
print("Corrélation de Spearman:", st.spearmanr(tot_achats_age.index, tot_achats_age.
print("Corrélation de Pearson : ",st.pearsonr(tot_achats_age.index,tot_achats_age.va
```

```
# expliquer les corrélations
# Graphique:
sns.scatterplot(data= tot_achats_age)
plt.ylabel("Montant en €")
plt.xlabel("Age en années")
plt.title("Montant des achats par âges")
plt.savefig("assets/graphiques/13.2 Montants des achats par âges.png")
```

```
Corrélation de Spearman: SpearmanrResult(correlation=-0.8576076555023923, pvalue=4.5
7972879340901e-23)
Corrélation de Pearson :  (-0.7788615764587009, 1.198396010000492e-16)
```



la fréquence d'achat

- le lien entre l'âge des clients et la fréquence d'achat
- Distribution empirique de la variable
  - modalité | effectif | frequence
    - les modalités : client_id, years, month | frequence : nb sessionid nunique
      - modalité : client_id | moyenne de session par mois

In [102…

```
# Fréquence d'achat en moyenne par mois
tmp = achats.groupby(["client_id","years","month"])["session_id"].nunique()
frequence_mensuelle= tmp.groupby("client_id").mean().reset_index()
print(tmp)
print(frequence_mensuelle)
# En moyenne le client c_1 fait 1.9 achat par mois
```

```
client_id  years  month
c_1        2021   6        1
                  7        4
                  8        1
                  9        2
                  10       1
                           ..
c_999      2022   10       1
                  11       2
                  12       1
           2023   1        4
                  2        1
Name: session_id, Length: 137134, dtype: int64
     client_id  session_id
0          c_1    1.941176
1         c_10    2.125000
```

```
2        c_100    1.000000
3       c_1000    4.227273
4       c_1001    2.350000
...        ...         ...
8595     c_995    1.125000
8596     c_996    3.434783
8597     c_997    1.263158
8598     c_998    1.642857
8599     c_999    2.210526

[8600 rows x 2 columns]
```

In [103…

```
# ajouter les ages au df : jointure outer avec customer
frequence_mensuelle = pd.merge(frequence_mensuelle,customers, on="client_id", how="o
frequence_mensuelle.head()
# on a un df client|moyenne fréquence achat| sex| naissance|age
```

Out[103…

	client_id	session_id	sex	birth	age
0	c_1	1.941176	m	1955	67
1	c_10	2.125000	m	1956	66
2	c_100	1.000000	m	1992	30
3	c_1000	4.227273	f	1966	56
4	c_1001	2.350000	m	1982	40

In [104…

```
frequence_mensuelle.isna().mean()*100
```

Out[104…

```
client_id     0.000000
session_id    0.243591
sex           0.000000
birth         0.000000
age           0.000000
dtype: float64
```

In [105…

```
# suppression des na
frequence_mensuelle = frequence_mensuelle.dropna() # ceux qui n'ont rien acheté
frequence_mensuelle.isna().mean()*100
```

Out[105…

```
client_id     0.0
session_id    0.0
sex           0.0
birth         0.0
age           0.0
dtype: float64
```

In [106…

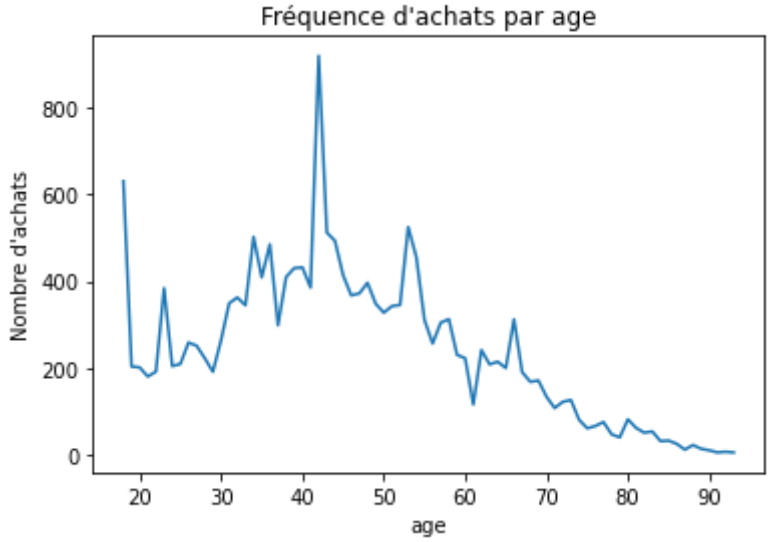
```
# Lien entre l'age et la fréquence achat mensuelle
frequence_age = frequence_mensuelle.groupby(['age'])['session_id'].sum()
print(round(frequence_age,2))
```

```
age
18    629.96
19    204.12
20    202.09
21    181.12
22    192.34
       ...
89     14.95
```

```
90     11.42
91      6.61
92      8.07
93      6.51
Name: session_id, Length: 76, dtype: float64
```

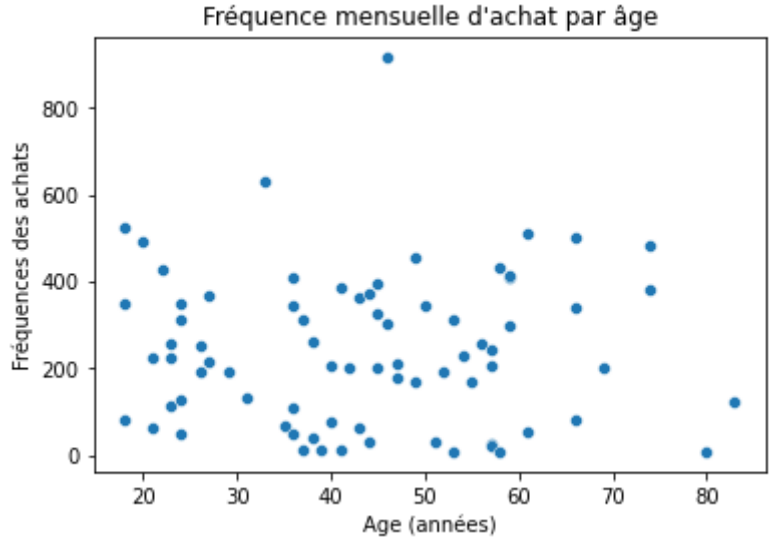
In [107…

```
frequence_age.plot()
plt.title("Fréquence d'achats par age")
plt.ylabel("Nombre d'achats")
plt.savefig("assets/graphiques/13.4 fequence_age.png")
```



In [108…

```
# Graphique
sns.scatterplot(x=frequence_mensuelle["age"], y=frequence_age)
plt.ylabel("Fréquences des achats")
plt.xlabel("Age (années)")
plt.title("Fréquence mensuelle d'achat par âge")
plt.savefig("assets/graphiques/13.3 Fréquences mensuelles achats par ages.png")
```



Coefficient de corrélation calculé sur les numéros d'ordre des valeurs des deux variables ordinales.

Le coefficient de corrélation sur les rangs (Rho de Spearman) s'interprète de la même manière qu'un coefficient de corrélation de Pearson : une valeur positive (maximum = +1) indique une

variation simultanée dans le même sens, une valeur négative (minimum = -1) une variation simultanée en sens inverse.

La différence entre les deux coefficients repose sur la nature des valeurs numériques. Le coefficient de Pearson est calculé à partir des données brutes des variables numériques. Le Rho de Spearman est calculé sur les rangs d'échelles ordinales.

In [109…

```
# Correlation entre l'age et le montant total des achats
# Test d'association entre deux variables quantitatives
print("Corrélation de Spearman:", st.spearmanr(frequence_mensuelle["age"], frequence
# La correlation n'est pas monotone elle n'est ni proche de 1 ni de -1 mais proche d
```

```
Corrélation de Spearman: SpearmanrResult(correlation=0.20589327229102006, pvalue=5.5
72887167216921e-83)
```

Analyse du lien entre age et catégorie par ANOVA

In [110…

```
achats.head()
```

Out[110…

date	index	id_prod	session_id	client_id	price	categ	sex	birth	age	years	mon
2021-03-01 00:01:07.843138	468405	0_1259	s_1	c_329	11.99	0.0	f	1967.0	55.0	2021	
2021-03-01 00:02:26.047414	497884	0_1390	s_2	c_664	19.37	0.0	m	1960.0	62.0	2021	
2021-03-01 00:02:38.311413	221213	0_1352	s_3	c_580	4.50	0.0	m	1988.0	34.0	2021	
2021-03-01 00:04:54.559692	316014	0_1458	s_4	c_7912	6.55	0.0	f	1989.0	33.0	2021	
2021-03-01 00:05:18.801198	595898	0_1358	s_5	c_2033	16.49	0.0	f	1956.0	66.0	2021	

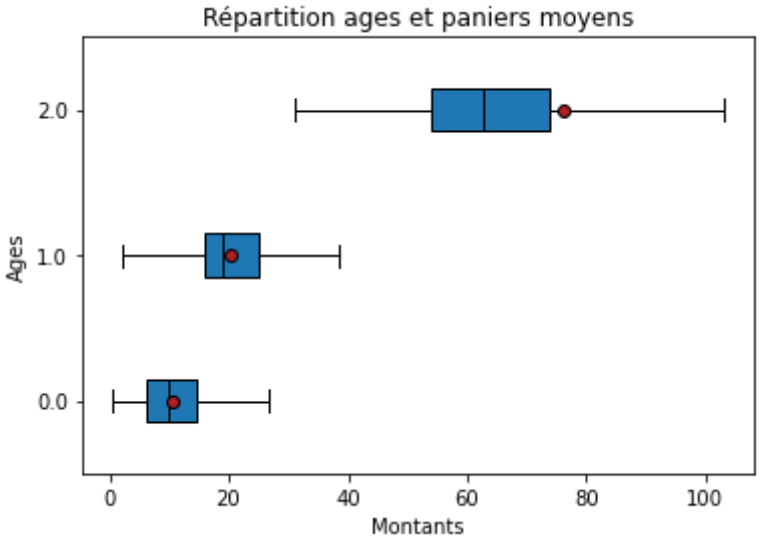
In [111…

```
X = "categ" # qualitative
Y = "price" # quantitative

sous_echantillon = achats
```

In [112…

```
modalites = sous_echantillon[X].unique()
groupes = []
for m in modalites:
    groupes.append(sous_echantillon[sous_echantillon[X]==m][Y])

# Propriétés graphiques (pas très importantes)
medianprops = {'color':"black"}
meanprops = {'marker':'o', 'markeredgecolor':'black',
            'markerfacecolor':'firebrick'}

plt.boxplot(groupes, labels=modalites, showfliers=False, medianprops=medianprops,
            vert=False, patch_artist=True, showmeans=True, meanprops=meanprops)
```

```
plt.title("Répartition ages et paniers moyens")
plt.xlabel("Montants")
plt.ylabel("Ages")
plt.savefig("assets/graphiques/16 lien entre age et panier moyen.png")
plt.show()
```



Les points rouges au milieu de chaque boîte à moustaches représentent la moyenne des valeurs.

On voit ici que les montants sont très différents d'une catégorie à l'autre. Vérifions maintenant cette affirmation par les chiffres, grâce à une modélisation.

Voici à présent le code permettant de calculer $\eta 2$

In [113…

```
X = "categ" # qualitative
Y = "price" # quantitative

def eta_squared(x,y):
    moyenne_y = y.mean()
    classes = []
    for classe in x.unique():
        yi_classe = y[x==classe]
        classes.append({'ni': len(yi_classe),
                        'moyenne_classe': yi_classe.mean()})
    SCT = sum([(yj-moyenne_y)**2 for yj in y])
    SCE = sum([c['ni']*(c['moyenne_classe']-moyenne_y)**2 for c in classes])
    return SCE/SCT

eta_squared(sous_echantillon[X],sous_echantillon[Y])
```

Out[113…

```
0.645765981330711
```

In [114…

```
# Forte corrélation entre prix et les ages
```

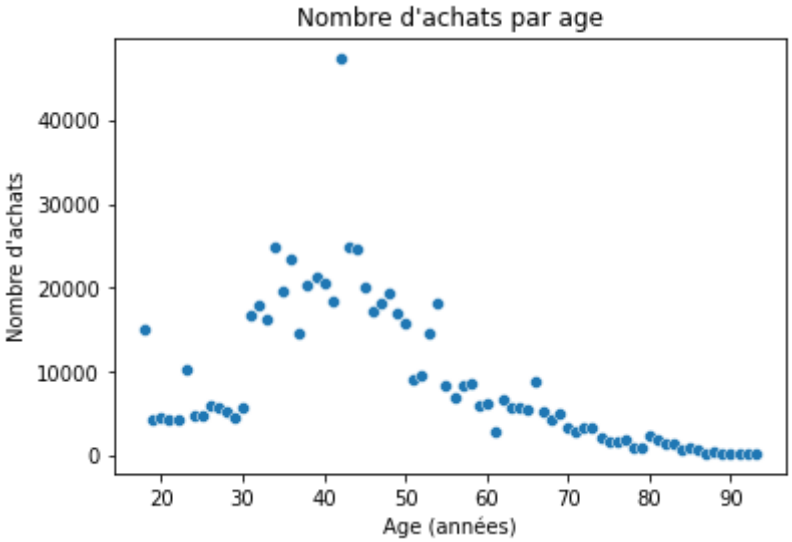
la taille du panier moyen et les catégories des livres achetés.

- et ensuite le lien entre l'âge des clients et la taille du panier moyen

In [115…

```
# Lien entre l'age et la taille du panier:
nbr_achats_age = achats.groupby(["age"])["session_id"].count()

# Correlation entre l'age et le montant total des achats:
print("Corrélation de Spearman:", st.spearmanr(nbr_achats_age.index,nbr_achats_age.v

# Graphique:
sns.scatterplot(data= nbr_achats_age)
plt.ylabel("Nombre d'achats")
plt.xlabel("Age (années)")
plt.title("Nombre d'achats par age")
plt.savefig("assets/graphiques/13.4 Nombre d'achats selon l'age.png")
```

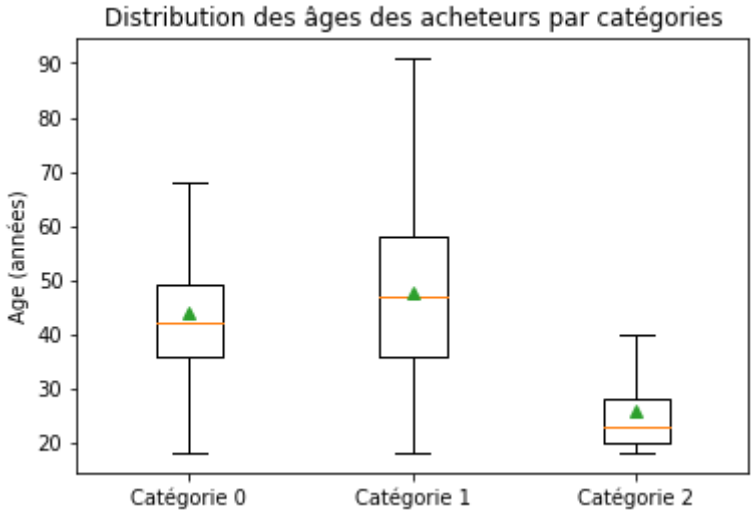
```
Corrélation de Spearman: SpearmanrResult(correlation=-0.6846206425153794, pvalue=9.1
52883867240306e-12)
```



le lien entre l'âge et les catégories des livres achetés.

In [116…

```
# Categories les plus achetees par age:
categ = achats["categ"].unique()
categ_age = {categ: achats.loc[achats["categ"]==categ, "age"] for categ in categ}

# Graphique:
plt.boxplot([
    categ_age[0],
    categ_age[1],
    categ_age[2]]
    ,labels=["Catégorie 0", "Catégorie 1", "Catégorie 2"],showmeans=True, showfliers
plt.ylabel("Age (années)")
plt.title("Distribution des âges des acheteurs par catégories")
plt.savefig("assets/graphiques/14.1 Distribution des ages des acheteurs par catégori
plt.show()
```



In [117…
```
# Lien entre age et categorie achetees:
#print(type(categ_age[0].values))
print(st.kruskal(categ_age[0], categ_age[1], categ_age[2]))
```

```
KruskalResult(statistic=79350.86927924873, pvalue=0.0)
```

Le test de Kruskal Wallis, est non paramétrique, les groupes doivent contenir plus de cinq individus. p-val < 0,01 **on rejette l'hypothèse nulle selon laquelle il n'y aurait pas de différence entre l'âge median d'achat par catégorie.** Kruskal est un test de rang.

- le **test des rangs** signés de Wilcoxon est une **alternative non-paramétrique au test de Student** pour des échantillons appariés. Le test s'intéresse à un paramètre de position : la médiane, le but étant de tester s'il existe un changement sur la médiane.
  - **Des échantillons appariés** sont des échantillons identiques, c'est à dire des échantillons **composés d'individus possédant les mêmes caractéristiques**. La ou les caractéristiques faisant l'objet de l'appariement peuvent être variables (âge, sexe, etc..).

La p-value nous indique que la probabilité de rejeter l'hypothèse nulle est inférieure à 0.0005. Dans ce cas, on peut rejeter l'hypothèse nulle H0 d'absence de différence significative entre les tranches ages. La catégorie 0 concerne en majorité les 20-70 La catégorie 1 touche l'ensemble de la clientèle La catégorie 2 n'est acheté que par les 20-40 ans

Lien entre le panier moyen et l'age

In [118…
```
panier_moyen = achats['price'].sum()/achats['session_id'].count()
panier_moyen
```

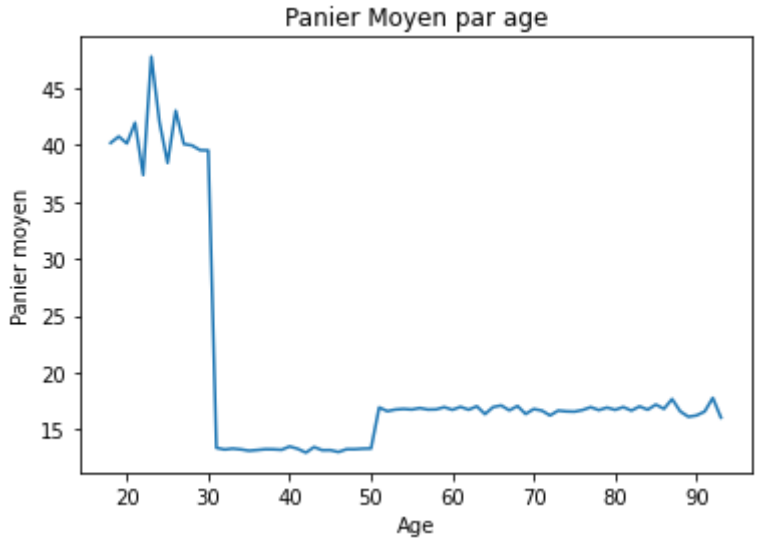
Out[118… `17.45234505367037`

In [119…
```
panierMoyen_age = achats.groupby(["age"]).agg({'price':'sum','session_id':'count'})
```

In [120…
```
panierMoyen_age["lien"] = panierMoyen_age["price"]/panierMoyen_age["session_id"]
panierMoyen_age.reset_index(inplace=True)
```

In [121…
```
# test correlation : corrélation linéaire te corrélarion de Pearson
plt.plot(panierMoyen_age['age'],panierMoyen_age['lien'])
```

```
plt.xlabel("Age")
plt.ylabel("Panier moyen")
plt.title("Panier Moyen par age")
plt.savefig("assets/graphiques/16 Panier moyen par age.png")
```



In [122…
```
panierMoyen_age.corr(method='pearson')
```

Out[122…

	age	price	session_id	lien
age	1.000000	-0.778862	-0.533705	-0.547907
price	-0.778862	1.000000	0.830352	0.249527
session_id	-0.533705	0.830352	1.000000	-0.267356
lien	-0.547907	0.249527	-0.267356	1.000000

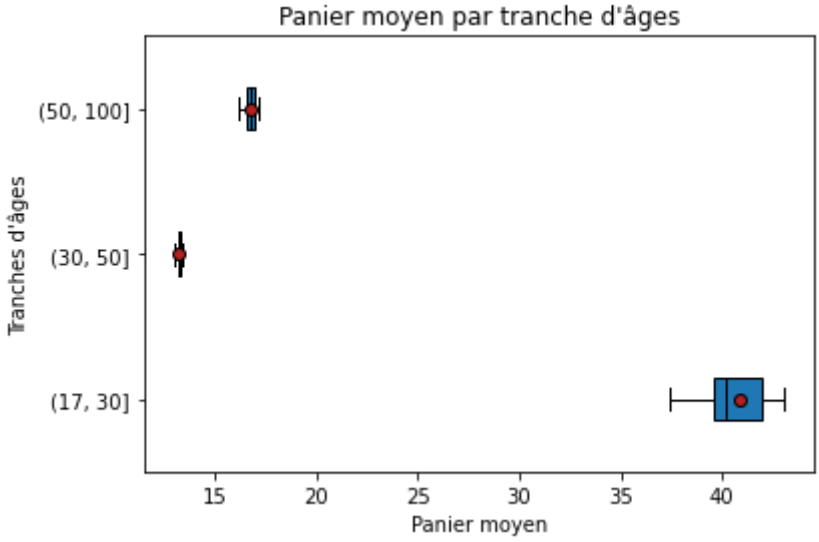
In [123…
```
# corresation négative => comportement par tranche d'ages => anova
```

In [124…
```
X = "age_rank" # qualitative
Y = "lien" # quantitative

sous_echantillon = panierMoyen_age
```

In [125…
```
sous_echantillon["age_rank"] = pd.cut(x= sous_echantillon["age"], bins=[17,30,50,100
```

In [126…
```
modalites = sous_echantillon[X].unique()
groupes = []
for m in modalites:
    groupes.append(sous_echantillon[sous_echantillon[X]==m][Y])

# Propriétés graphiques (pas très importantes)
medianprops = {'color':"black"}
meanprops = {'marker':'o', 'markeredgecolor':'black',
            'markerfacecolor':'firebrick'}

plt.boxplot(groupes, labels=modalites, showfliers=False, medianprops=medianprops,
            vert=False, patch_artist=True, showmeans=True, meanprops=meanprops)
plt.title("Panier moyen par tranche d'âges")
plt.xlabel("Panier moyen")
```

```
plt.ylabel("Tranches d'âges")
plt.savefig("assets/graphiques/17 Panier moyen par tranche age.png")
plt.show()
```



In [127…

```
X = "age_rank" # qualitative
Y = "lien" # quantitative

def eta_squared(x,y):
    moyenne_y = y.mean()
    classes = []
    for classe in x.unique():
        yi_classe = y[x==classe]
        classes.append({'ni': len(yi_classe),
                        'moyenne_classe': yi_classe.mean()})
    SCT = sum([(yj-moyenne_y)**2 for yj in y])
    SCE = sum([c['ni']*(c['moyenne_classe']-moyenne_y)**2 for c in classes])
    return SCE/SCT

eta_squared(sous_echantillon[X],sous_echantillon[Y])
```

Out[127…

```
0.9881431123434874
```

In [128…

```
#forte correlation en fonction des tranches d'age choisies
```