Mendes_Elodie_1_notebook_072025

November 22, 2025

PROJET 9 - ANALYSE DES VENTES D'UNE LIBRAIRIE EN LIGNE

Dans ce notebook, nous allons analyser les ventes et le comportement des clients sur le site internet sur 2 ans d'activité .

Nous allons suivre les étapes suivantes:

Etape 1. Importation des librairies et chargement des fichiers

Etape 2. Explorations des données

Etape 3. Analyse des indicateurs de ventes

Etape 4. Analyse des corrélations

Recommandations

Etape 1. Importation des librairies et chargement des fichiers

0.1 1.1 - Importation des librairies

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from wordcloud import WordCloud
from scipy.stats import norm
from scipy.stats import shapiro
from scipy.stats import spearmanr
from scipy.stats import chi2_contingency
from scipy.stats import kruskal
import scipy.stats as stats
from scipy.stats import normaltest
#from ydata_profiling import ProfileReport
```

0.2 1.2 - Chargement des fichiers

[2]:
```
#Importation du fichier client
clients = pd.read_csv ('customers.csv', sep=";")

#Importation du fichier produit
```

```
produits = pd.read_csv ('products.csv', sep=";")

#Importation du fichier des transactions
transactions = pd.read_csv ('Transactions.csv', sep=";")
```

Etape 2. Analyse Exploratoire des données

0.3 2.1 - Fichier clients

[3]:
```
#Description du dataframe
clients.info()
#Affichage du dataframe
clients.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8621 entries, 0 to 8620
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   client_id  8621 non-null   object
 1   sex        8621 non-null   object
 2   birth      8621 non-null   int64 
dtypes: int64(1), object(2)
memory usage: 202.2+ KB
```

[3]:

	client_id	sex	birth
0	c_4410	f	1967
1	c_7839	f	1975
2	c_1699	f	1984
3	c_5961	f	1962
4	c_5320	m	1943

Il n' y a aucune valeur manquante sur le fichier clients

[4]:
```
#Identifier la longueur des identifiants clients
clients['client_id'].str.len().unique()
```

[4]:
```
array([6, 5, 4, 3])
```

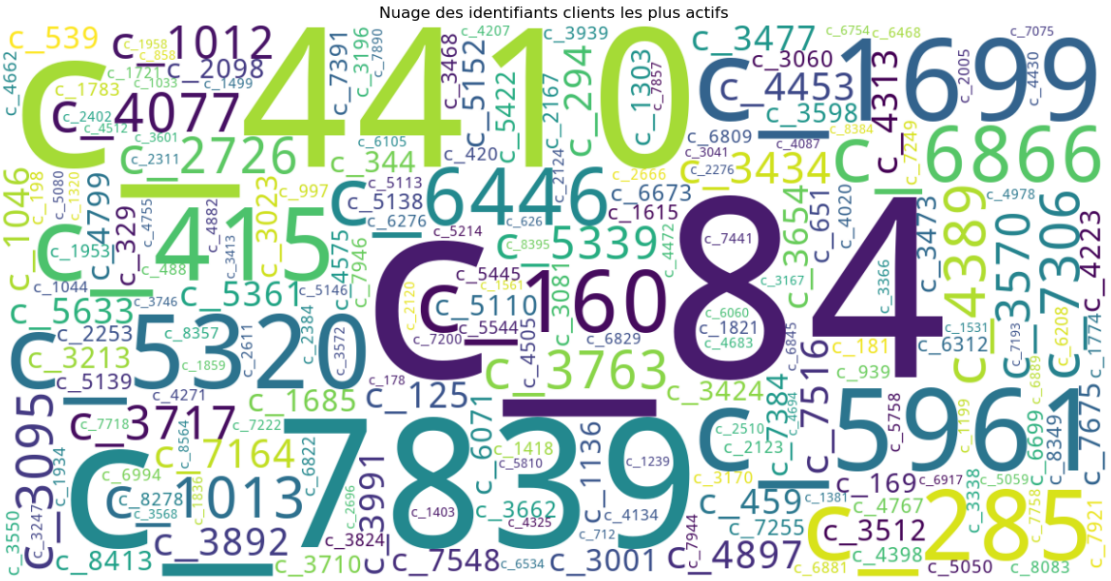
Nous avons des identifiants clients de 3 à 6 caractères.

[5]:
```
clients['client_id'].unique()
```

[5]:
```
array(['c_4410', 'c_7839', 'c_1699', …, 'c_5119', 'c_5643', 'c_84'],
      dtype=object)
```

[6]:
```
# création de la variable qui créer un dictionnaire d'identifiants clients
client_freq = clients['client_id'].value_counts().to_dict()

# Créer le wordcloud
```

```
wc = WordCloud(width=1200, height=600, background_color='white',␣
 ↪collocations=False)
wc.generate_from_frequencies(client_freq)

# Affichage
plt.figure(figsize=(15,8))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.title('Nuage des identifiants clients les plus actifs')
plt.show()
```



Il semble que le format des identifiants suit le format suivant '*C_+chiffres*'

[7]:
```
#1. Création d'une variable qui va déterminer le format à avoir
identifiant = r'^c_\d+$'

#2. Création de colonne pour identifier les identifiants qui matchent avec le␣
 ↪format standard
clients['identifiant_ok'] = clients['client_id'].str.match(identifiant)

#2. Affichage des idenfiants anormaux
clients[~clients['identifiant_ok']]
```

[7]:
```
Empty DataFrame
Columns: [client_id, sex, birth, identifiant_ok]
Index: []
```

Il n' y a donc aucun identifiant client faux

[8]:
```
#Identifier le nombre de genre pour remarquer des erreurs
print(f"il y a 2 genres {clients['sex'].unique()} féminin/masculin.")
```

```
il y a 2 genres ['f' 'm'] féminin/masculin.
```

[9]:
```
#Suppression de la colonne *identifiant_ok*
clients.drop(columns=['identifiant_ok'],inplace=True)
```

[10]:
```
#Identifier les dates de naissance aberrantes
clients['birth'].sort_values(ascending=True).unique()
```

[10]:
```
array([1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939,
       1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950,
       1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961,
       1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972,
       1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983,
       1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994,
       1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004])
```

Il n'y a pas d'année de naissance atypique. Je peux calculer l'âge du client à partir de sa date de naissance.

[11]:
```
#Calcul de l'âge (en année révolue) à partir de l'année de naissance

 # 1.Récupère l'année actuelle
annee_actuelle = pd.to_datetime('today').year

# 2. Création d'une nouvelle colonne âge
clients['âge'] = annee_actuelle - (clients['birth'])

#Affichage du dataset avec la nouvelle colonne
clients
```

[11]:

	client_id	sex	birth	âge
0	c_4410	f	1967	58
1	c_7839	f	1975	50
2	c_1699	f	1984	41
3	c_5961	f	1962	63
4	c_5320	m	1943	82
…	…	..	…	…
8616	c_7920	m	1956	69
8617	c_7403	f	1970	55
8618	c_5119	m	1974	51
8619	c_5643	f	1968	57
8620	c_84	f	1982	43

[8621 rows x 4 columns]

[12]:
```
#Création des tranches d'âge

#1. Etablir une liste de tranche
bins = [18, 25, 35, 45, 55, 65, 80, 100]
labels = ['18-25 ans','26-35 ans','36-45 ans','46-55 ans','56-65 ans','66-80␣
 ↪ans','81 ans et +']

#2. Création de la colonne 'tranche d'âge'
clients['tranche_age'] = pd.cut(clients['âge'], bins=bins, labels=labels,␣
 ↪right=True)
```

[13]:
```
#Description statistique du dataset
clients.describe(include='all')
```

[13]:

	client_id	sex	birth	âge	tranche_age
count	8621	8621	8621.000000	8621.000000	8621
unique	8621	2	NaN	NaN	7
top	c_84	f	NaN	NaN	36-45 ans
freq	1	4490	NaN	NaN	1713
mean	NaN	NaN	1978.275606	46.724394	NaN
std	NaN	NaN	16.917958	16.917958	NaN
min	NaN	NaN	1929.000000	21.000000	NaN
25%	NaN	NaN	1966.000000	33.000000	NaN
50%	NaN	NaN	1979.000000	46.000000	NaN
75%	NaN	NaN	1992.000000	59.000000	NaN
max	NaN	NaN	2004.000000	96.000000	NaN

Il n' y a pas de doublons.

Il y a une majorité de femme parmi la clientèle.

La moyenne d'âge est de 47 ans.

0.4 2.2 - Fichier produits

[14]:
```
#Description du dataframe
produits.info()

#Affichage du dataframe
produits.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3286 entries, 0 to 3285
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id_prod  3286 non-null   object
 1   price    3286 non-null   float64
 2   categ    3286 non-null   int64
```

```
dtypes: float64(1), int64(1), object(1)
memory usage: 77.1+ KB
```

[14]:

	id_prod	price	categ
0	0_1421	19.99	0
1	0_1368	5.13	0
2	0_731	17.99	0
3	1_587	4.99	1
4	0_1507	3.99	0

Il n'y a pas de valeurs manquantes.

[15]:
```
#Repérer il y a combien de catégorie de produit
produits['categ'].unique()
```

[15]:
```
array([0, 1, 2])
```

Il y a 3 catégories de produit: "0", "1" et "2" .

[16]:
```
# Création de la variable qui créer un dictionnaire d'identifiants produits
prod_freq = produits['id_prod'].value_counts().to_dict()

# Créer le wordcloud
wc = WordCloud(width=1200, height=600, background_color='white',␣
 ↪collocations=False)
wc.generate_from_frequencies(prod_freq)

# Affichage
plt.figure(figsize=(15,8))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.title('Nuage des identifiants produits produits les plus actifs')
plt.show()
```

[17]:
```
#Vérifier si il n' y a pas des prix négatifs
(produits['price']< 0).sum()
```

[17]:
```
np.int64(0)
```

Il n' y a aucun prix négatifs

[18]:
```
#Description statistique des prix et des catégories de produit
produits.describe(include ="all")
```

[18]:

	id_prod	price	categ
count	3286	3286.000000	3286.000000
unique	3286	NaN	NaN
top	0_1920	NaN	NaN
freq	1	NaN	NaN
mean	NaN	21.863597	0.370359
std	NaN	29.849786	0.615446
min	NaN	0.620000	0.000000
25%	NaN	6.990000	0.000000
50%	NaN	13.075000	0.000000
75%	NaN	22.990000	1.000000
max	NaN	300.000000	2.000000

Il n'y a pas de doublons.

Le prix moyen d'un livre est de 22€.

0.5 2.3 - Fichier transactions

[19]:
```
#Description du dataframe
transactions.info()

#Affichage du dataframe
transactions.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 687534 entries, 0 to 687533
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   id_prod     687534 non-null  object
 1   date        687534 non-null  object
 2   session_id  687534 non-null  object
 3   client_id   687534 non-null  object
dtypes: object(4)
memory usage: 21.0+ MB
```

[19]:

	id_prod	date	session_id	client_id
0	0_1259	2021-03-01 00:01:07.843138	s_1	c_329
1	0_1390	2021-03-01 00:02:26.047414	s_2	c_664
2	0_1352	2021-03-01 00:02:38.311413	s_3	c_580
3	0_1458	2021-03-01 00:04:54.559692	s_4	c_7912
4	0_1358	2021-03-01 00:05:18.801198	s_5	c_2033

[20]:
```
#Conversion de la colonne 'date' en type datetime
transactions['date'] = pd.to_datetime(transactions['date'], errors='coerce')
print(transactions['date'].dtype)
```

```
datetime64[ns]
```

La colonne "*date*" a été converti sous le format date

[21]:
```
#Affichage de tous les mois de transactions présents dans le dataset des␣
 ↪transactions
transactions['date'].dt.to_period('M').sort_values(ascending= True).unique()
```

[21]:
```
<PeriodArray>
['2021-03', '2021-04', '2021-05', '2021-06', '2021-07', '2021-08', '2021-09',
 '2021-10', '2021-11', '2021-12', '2022-01', '2022-02', '2022-03', '2022-04',
 '2022-05', '2022-06', '2022-07', '2022-08', '2022-09', '2022-10', '2022-11',
 '2022-12', '2023-01', '2023-02']
Length: 24, dtype: period[M]
```

Nous avons des transactions allant de mars 2021 à Février 2023 sans dates manquantes.

[22]:
```
#Identifier s'il y a plusieurs fois la même session d'achat
transactions[transactions['session_id'].duplicated() == True]
```

[22]:
```
       id_prod                       date session_id client_id
7        0_279 2021-03-01 00:07:48.507530        s_6    c_4908
11      0_1638 2021-03-01 00:10:37.223732        s_3     c_580
13      0_1159 2021-03-01 00:11:57.832228        s_7    c_1609
15      0_1475 2021-03-01 00:16:16.649539        s_6    c_4908
16      0_2056 2021-03-01 00:16:49.525524       s_12    c_2505
…         …                          …         …       …
687523  0_1435 2023-02-28 23:36:25.400073   s_348428    c_7481
687524  0_1039 2023-02-28 23:43:05.079569   s_348438    c_7144
687525   1_417 2023-02-28 23:45:54.817107   s_348438    c_7144
687527   0_998 2023-02-28 23:47:05.145663   s_348427    c_4476
687533  0_1398 2023-02-28 23:58:30.792755   s_348435    c_3575

[342029 rows x 4 columns]
```

[23]:
```
#Description statistique du fichier transactions
transactions.describe(include="all")
```

[23]:
```
       id_prod                          date session_id client_id
count   687534                        687534     687534    687534
unique    3265                           NaN     345505      8600
top      1_369                           NaN   s_118668    c_1609
freq      2340                           NaN         14     25586
mean       NaN  2022-03-01 21:24:00.618519296        NaN       NaN
min        NaN     2021-03-01 00:01:07.843138        NaN       NaN
25%        NaN  2021-09-10 10:35:20.642323456        NaN       NaN
50%        NaN  2022-02-27 06:50:25.400120064        NaN       NaN
75%        NaN  2022-08-28 22:16:49.841665536        NaN       NaN
max        NaN     2023-02-28 23:58:30.792755        NaN       NaN
```

La librairie offre 3286 produits sur son site mais ici on voit qu'il y a 21 produits non vendus ou bien 21 clients qui n'ont jamais commandé (8621 clients uniques au total).

1 Etape 3. Jointure des fichiers

1.1 3.1 Jointure entre le fichier produit et transactions

[24]:
```
#Fusion du fichier 'produit' et le fichier 'transaction'
df_ventes = pd.merge(produits,transactions, on= "id_prod", how= "outer")
df_ventes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 687555 entries, 0 to 687554
Data columns (total 6 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id_prod     687555 non-null  object
 1   price       687555 non-null  float64
```

```
 2   categ       687555 non-null  int64
 3   date        687534 non-null  datetime64[ns]
 4   session_id  687534 non-null  object
 5   client_id   687534 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(1), object(3)
memory usage: 31.5+ MB
```

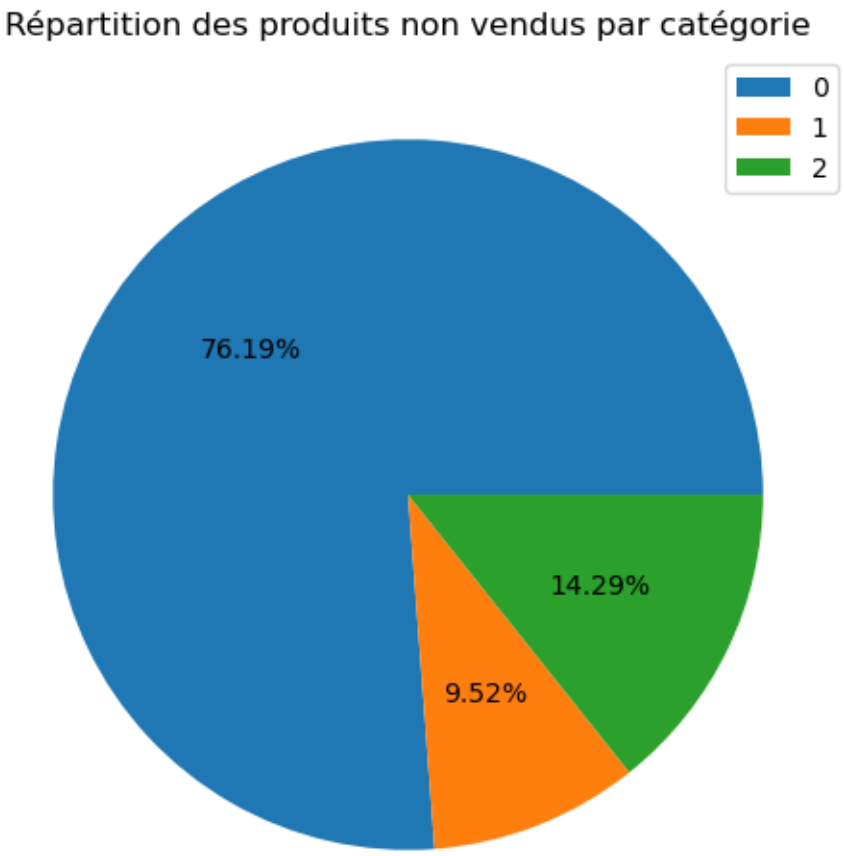
Il manque des dates de transactions, des identifiants de sessions et des identifiants clients

[25]:
```
#Vérification si il y a des livres qui n'ont jamais été vendus
df_ventes = pd.merge(produits,transactions, on= "id_prod", how=␣
 ↪"outer",indicator='_merge')
print(f"Nombre de produits jamais vendus : {df_ventes[df_ventes['_merge'] ==␣
 ↪'left_only'].shape[0]}")

#Part des produits non vendus sur le total de livres proposés
print(f"Les produits non vendus représentent {round((21/produits.
 ↪shape[0])*100,2)}% de l'offre total")
```

```
Nombre de produits jamais vendus : 21
Les produits non vendus représentent 0.64% de l'offre total
```

Lorsque je joins les données de transactions et ceux des produits, je remarque qu'il y a 21 livres de mon offre qui n'ont pas de date de transaction donc qui n'ont pas été achetés.

[26]:
```
# Visualisation de la Répartition des 21 produits non vendus


produit_non_vendus = df_ventes[df_ventes['date'].isna()]
produit_non_vendus_categ = df_ventes[df_ventes['date'].isna()].groupby('categ')␣
 ↪['id_prod'].count().reset_index()

produit_non_vendus_categ['categ'] = produit_non_vendus_categ['categ'].
 ↪astype(str)

#Visualisation
plt.figure(figsize=(10,6))
plt.pie(
    data= produit_non_vendus_categ,
    x='id_prod',
    autopct='%1.2f%%'
)
plt.title("Répartition des produits non vendus par catégorie")
plt.legend(produit_non_vendus_categ['categ'])
plt.show()
```



Sur les 21 produits jamais vendus, 60% proviennent de la catégorie 0 dont les prix sont pourtant les plus bas de l'offre.

[27]:
```
produit_non_vendus_categ
```

[27]:

	categ	id_prod
0	0	16
1	1	2
2	2	3

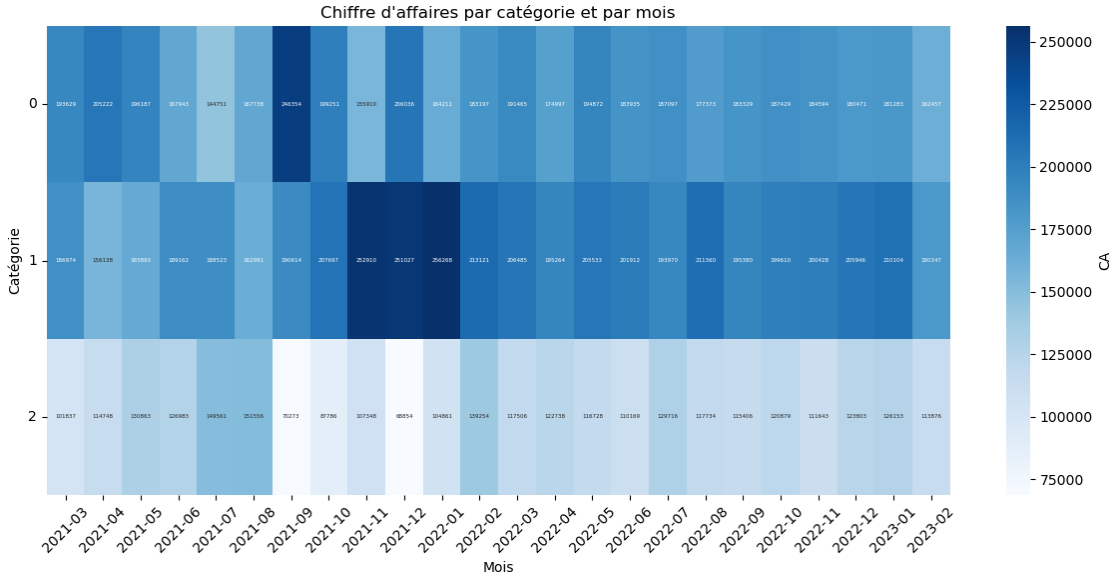
[28]:
```
#Analyse univariée des prix
px.box(df_ventes,
       x='price',
       title= "Boxplot des prix de vente")
```

Il y a énormément de prix qui sortent du lot. Il se peut qu'il y ait des ouvrages rares, des collections rares qui justifient ses prix.

[29]:
```
#Conversion de la colonne date du dataframe 'df_ventes' en mois
df_ventes['mois'] = df_ventes['date'].dt.to_period('M')

#A partir de la jointure externe, je groupe le CA par mois et par catégorie
ca_cat_mensuel = df_ventes.groupby(['categ', 'mois'])['price'].sum().
 ↪unstack(fill_value=0)
ca_cat_mensuel
```

[29]:
```
mois     2021-03    2021-04    2021-05    2021-06    2021-07    2021-08  \
categ
0      193629.17  205222.46  196186.72  167943.15  144750.79  167737.62
1      186974.17  156138.35  165893.40  189162.04  188523.27  162991.38
2      101837.27  114748.49  130863.35  126983.37  149561.34  151555.79

mois     2021-09    2021-10    2021-11    2021-12  …    2022-05    2022-06  \
categ                                               …
0      246353.91  199250.83  155909.56  206036.24  …  194872.34  183934.86
1      190613.78  207696.74  252910.39  251026.75  …  205532.63  201912.06
2       70272.99   87785.59  107347.78   68854.29  …  116727.63  110169.20

mois     2022-07    2022-08    2022-09    2022-10    2022-11    2022-12  \
categ
0      187097.00  177372.76  183329.24  187429.17  184594.35  180470.70
1      193969.72  211360.09  195379.54  199609.66  200427.99  205945.71
2      129716.40  117734.42  115405.75  120878.94  111642.60  123803.09

mois     2023-01    2023-02
categ
0      181283.06  162457.00
1      210104.41  180347.24
2      126153.08  113875.52

[3 rows x 24 columns]
```

[30]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.heatmap(ca_cat_mensuel, annot=True, fmt='.0f', cmap='Blues',␣
 ↪cbar_kws={'label': 'CA'},annot_kws={"size":4})
plt.title("Chiffre d'affaires par catégorie et par mois")
plt.xlabel("Mois")
plt.ylabel("Catégorie")
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



Il n' y a pas de mois sans ventes

1.2 3.2 Jointure entre le fichier transactions et clients

Je joins les 2 fichiers pour identifier si il y a des clients qui n'ont jamais achetés ou des transactions qui ne sont lié à aucun client

[31]:
```
#Fusion du fichier 'transactions' et le fichier 'clients'
df_achat_client = pd.merge(transactions,clients, on= "client_id", how= "outer")
df_achat_client.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 687555 entries, 0 to 687554
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   id_prod      687534 non-null  object
 1   date         687534 non-null  datetime64[ns]
 2   session_id   687534 non-null  object
 3   client_id    687555 non-null  object
 4   sex          687555 non-null  object
 5   birth        687555 non-null  int64
 6   âge          687555 non-null  int64
 7   tranche_age  687555 non-null  category
dtypes: category(1), datetime64[ns](1), int64(2), object(4)
memory usage: 37.4+ MB
```

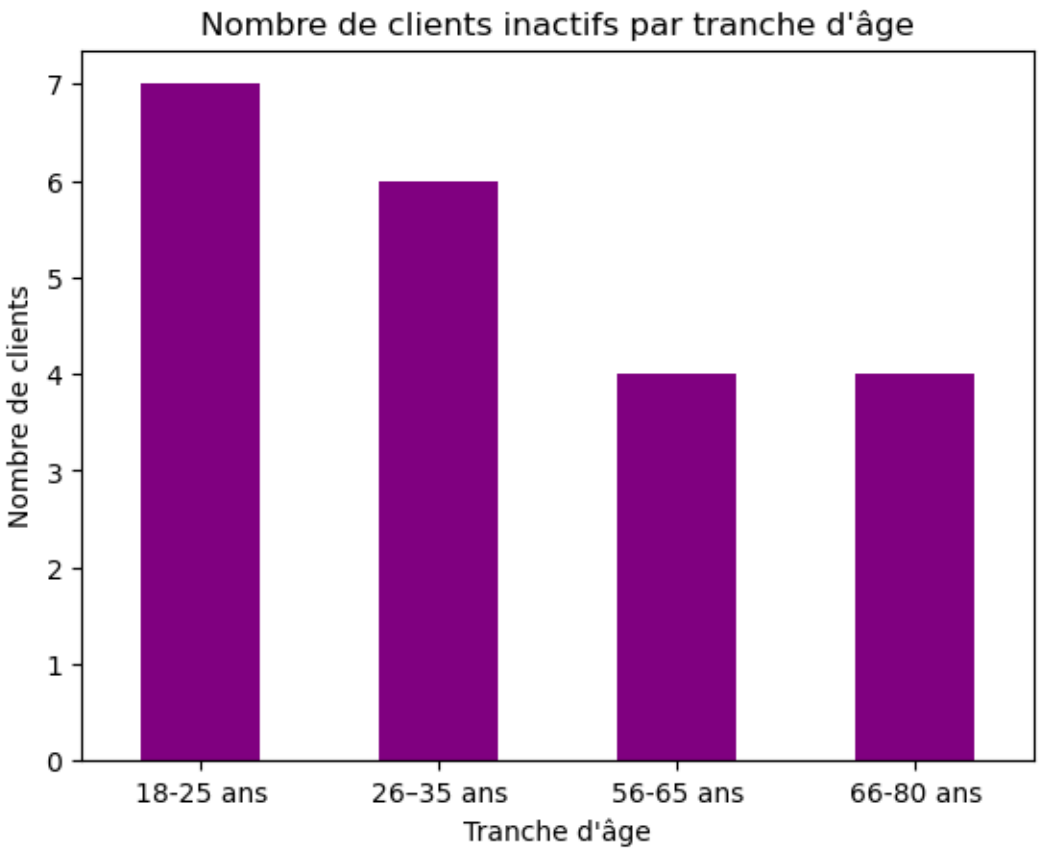
[32]:
```
df_achat_client = pd.merge(transactions,clients, on= "client_id", how=␣
 ↪"outer",indicator='_merge')
```

```
print(f"Nombre de clients qui n'ont jamais commandé:␣
  ↪{df_achat_client[df_achat_client['_merge'] == 'right_only'].shape[0]}")

client_inactif = df_achat_client[df_achat_client['_merge'] == 'right_only']
print(f"Les clients inactifs représentent {round((client_inactif.shape[0]*100/
  ↪clients.shape[0]),2)}% des clients")
```

```
Nombre de clients qui n'ont jamais commandé: 21
Les clients inactifs représentent 0.24% des clients
```

[33]:
```
client_inactif.groupby('tranche_age',observed='True') ['client_id'].count().
  ↪plot(
    kind='bar',
    x='tranche_age',
    y='client_id',
    title="Nombre de clients inactifs par tranche d'âge", color='purple')
plt.xlabel("Tranche d'âge")
plt.xticks(rotation=0)
plt.ylabel("Nombre de clients")
```

[33]:
```
Text(0, 0.5, 'Nombre de clients')
```



[34]:
```
#Affichage des mois présents après la jointure
df_achat_client['mois'] = df_achat_client['date'].dt.to_period('M')

#Affichage des mois présents après la jointure
df_achat_client['mois'].sort_values(ascending=True).unique()
```

[34]:
```
<PeriodArray>
['2021-03', '2021-04', '2021-05', '2021-06', '2021-07', '2021-08', '2021-09',
 '2021-10', '2021-11', '2021-12', '2022-01', '2022-02', '2022-03', '2022-04',
 '2022-05', '2022-06', '2022-07', '2022-08', '2022-09', '2022-10', '2022-11',
 '2022-12', '2023-01', '2023-02',     'NaT']
Length: 25, dtype: period[M]
```

1.3 3.3 Jointure interne des 3 fichiers

Je vais créer un dataframe qui va regrouper toutes les transactions entre les produits vendus et les clients qui ont achetés qui va servir de base pour l'analyse des ventes.

[35]:
```
df_merge= pd.merge(df_ventes, clients, on='client_id',how= 'inner')
df_merge.info()
df_merge.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 687534 entries, 0 to 687533
Data columns (total 12 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   id_prod      687534 non-null  object
 1   price        687534 non-null  float64
 2   categ        687534 non-null  int64
 3   date         687534 non-null  datetime64[ns]
 4   session_id   687534 non-null  object
 5   client_id    687534 non-null  object
 6   _merge       687534 non-null  category
 7   mois         687534 non-null  period[M]
 8   sex          687534 non-null  object
 9   birth        687534 non-null  int64
 10  âge          687534 non-null  int64
 11  tranche_age  687534 non-null  category
dtypes: category(2), datetime64[ns](1), float64(1), int64(3), object(4),
period[M](1)
memory usage: 53.8+ MB
```

[35]:

	id_prod	price	categ	date	session_id	client_id	\
0	0_0	3.75	0	2021-03-01 13:45:51.575117	s_282	c_5152	
1	0_0	3.75	0	2021-03-02 06:42:55.351333	s_621	c_2917	

```
2     0_0   3.75      0 2021-03-02 18:49:49.651862      s_852    c_3988
3     0_0   3.75      0 2021-03-02 21:57:33.862118      s_908    c_1004
4     0_0   3.75      0 2021-03-03 23:59:48.997483     s_1379     c_278

  _merge     mois sex  birth  âge tranche_age
0   both  2021-03   f   1986   39   36-45 ans
1   both  2021-03   m   1988   37   36-45 ans
2   both  2021-03   f   1962   63   56-65 ans
3   both  2021-03   m   1973   52   46-55 ans
4   both  2021-03   f   1987   38   36-45 ans
```

[36]:
```
df_merge.describe(include='all')
```

[36]:
```
        id_prod          price          categ                           date  \
count    687534  687534.000000  687534.000000                         687534
unique     3265            NaN            NaN                            NaN
top       1_369            NaN            NaN                            NaN
freq       2340            NaN            NaN                            NaN
mean        NaN      17.493918       0.448789  2022-03-01 21:24:00.618518784
min         NaN       0.620000       0.000000     2021-03-01 00:01:07.843138
25%         NaN       8.990000       0.000000  2021-09-10 10:35:20.642323456
50%         NaN      13.990000       0.000000  2022-02-27 06:50:25.400120064
75%         NaN      19.080000       1.000000  2022-08-28 22:16:49.841665536
max         NaN     300.000000       2.000000     2023-02-28 23:58:30.792755
std         NaN      18.238337       0.594563                            NaN

       session_id client_id  _merge     mois     sex          birth  \
count      687534    687534  687534   687534  687534  687534.000000
unique     345505      8600       1       24       2            NaN
top      s_118668    c_1609    both  2021-09       m            NaN
freq           14     25586  687534    33314  344841            NaN
mean          NaN       NaN     NaN      NaN     NaN    1977.817391
min           NaN       NaN     NaN      NaN     NaN    1929.000000
25%           NaN       NaN     NaN      NaN     NaN    1970.000000
50%           NaN       NaN     NaN      NaN     NaN    1980.000000
75%           NaN       NaN     NaN      NaN     NaN    1987.000000
max           NaN       NaN     NaN      NaN     NaN    2004.000000
std           NaN       NaN     NaN      NaN     NaN      13.607935

                 âge tranche_age
count   687534.000000      687534
unique            NaN           7
top               NaN   36-45 ans
freq              NaN      229244
mean        47.182609         NaN
min         21.000000         NaN
25%         38.000000         NaN
```

```
50%          45.000000          NaN
75%          55.000000          NaN
max          96.000000          NaN
std          13.607935          NaN
```

[37]:
```
#Création d'une clé primaire en auto_=_incrémentation pour pouvoir exporter le dataset.
df_merge['transaction_id'] = range(1, len(df_merge) + 1)
```

[38]:
```
#Affichage du dataset fusionné
df_merge.head()
```

[38]:

	id_prod	price	categ	date	session_id	client_id	_merge	mois	sex	birth	âge	tranche_age	transaction_id
0	0_0	3.75	0	2021-03-01 13:45:51.575117	s_282	c_5152	both	2021-03	f	1986	39	36-45 ans	1
1	0_0	3.75	0	2021-03-02 06:42:55.351333	s_621	c_2917	both	2021-03	m	1988	37	36-45 ans	2
2	0_0	3.75	0	2021-03-02 18:49:49.651862	s_852	c_3988	both	2021-03	f	1962	63	56-65 ans	3
3	0_0	3.75	0	2021-03-02 21:57:33.862118	s_908	c_1004	both	2021-03	m	1973	52	46-55 ans	4
4	0_0	3.75	0	2021-03-03 23:59:48.997483	s_1379	c_278	both	2021-03	f	1987	38	36-45 ans	5

[39]:
```
#Affichage des mois présents après la jointure
date= df_merge['date'].dt.to_period('M').sort_values(ascending= True).unique()
date
```

[39]:
```
<PeriodArray>
['2021-03', '2021-04', '2021-05', '2021-06', '2021-07', '2021-08', '2021-09',
 '2021-10', '2021-11', '2021-12', '2022-01', '2022-02', '2022-03', '2022-04',
 '2022-05', '2022-06', '2022-07', '2022-08', '2022-09', '2022-10', '2022-11',
 '2022-12', '2023-01', '2023-02']
Length: 24, dtype: period[M]
```

[40]:
```
#Vérification des valeurs manquantes
df_merge.isnull().sum()
```

[40]:
```
id_prod           0
price             0
categ             0
date              0
session_id        0
client_id         0
_merge            0
```

```
mois              0
sex               0
birth             0
âge               0
tranche_age       0
transaction_id    0
dtype: int64
```

Il n'y a aucune valeurs manquantes dans le nouveau dataset fusionné

[41]:
```
#Graphique des prix de vente par catégorie

fig = px.box(df_merge, x= 'categ', y= 'price', labels={'categ':'Catégories de␣
 ↪livres','price':'Prix de vente'},title="Boxplot des prix de vente par␣
 ↪catégorie", color='categ')
fig.show()
```

On remarque que chaque catégorie de livre a une gamme de prix différente. Les livres les plus chers proviennent de la catégorie 2 et la moins chère sont la catégorie 0.

Dans chaque catégorie, on observe des prix sortant du lot surtout pour la catégorie 2, où le nombre d'articles avec des prix élevés est le plus important. Il doit y avoir des livres d'exceptions (nous n'avons pas les titre de livre pour pouvoir confirmer).

Etape 3. Analyse des indicateurs de ventes

1.4 3.1 Chiffre d'affaires par client

[42]:
```
client_ca = df_merge.groupby(['client_id']).agg({'price':'sum','tranche_age':
 ↪'first','sex':'first','session_id':'nunique'}).reset_index()

#Visualisation
px.box( client_ca,
       x= 'price',
      title="Boxplot du chiffre d'affaires par client",
      labels={'price':"Chiffre d'affaires (€)"})
```

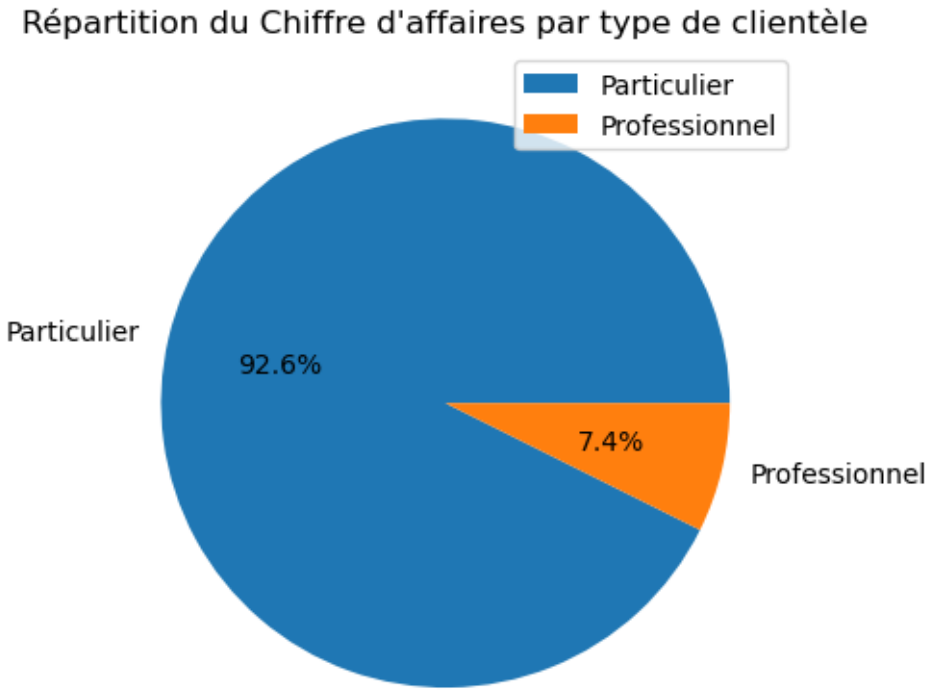
Il y a clairement 4 clients qui se démarquent avec un chiffre d'affaires très élevé (au dessus de 100k de CA). Cela révèle que l'on a des différents type de clientèle, une clientèle BtoB minoritaire et les particuliers. Nous allons voir le répartition au sein de la clientèle.

[43]:
```
#Création de la colonne 'type de client' sur le dataset df_merge.
client_ca['type_client'] = np.where(client_ca['price'] > 100000,␣
 ↪'Professionnel','Particulier')
```

[44]:
```
Repartition_ca_type_client = client_ca.groupby('type_client') ['price'].sum().
 ↪reset_index()

#Visualisation
```

```
plt.
  ↪pie(x=Repartition_ca_type_client['price'],labels=Repartition_ca_type_client['type_client'],
  ↪autopct= '%1.1f%%')
plt.title("Répartition du Chiffre d'affaires par type de clientèle")
plt.legend()
```

[44]: <matplotlib.legend.Legend at 0x22b09d37d90>



Près de 93% du chiffre d'affaires proviennent des clients particuliers contre 7.4% des professionnels.

[45]:
```
#Création dataframe des clients pros
client_pro = client_ca[client_ca['type_client'] == 'Professionnel']
client_pro
```

[45]:

	client_id	price	tranche_age	sex	session_id	type_client
677	c_1609	326039.89	36-45 ans	m	10997	Professionnel
2724	c_3454	114110.57	56-65 ans	m	5571	Professionnel
4388	c_4958	290227.03	26-35 ans	m	3851	Professionnel
6337	c_6714	153918.60	56-65 ans	f	2620	Professionnel

1.5 3.2 Exclusion des clients atypiques (BtoB) du dataset

[46]:
```
#Exclusion des 4 clients BtoB
client_atypique = ['c_1609','c_4958','c_6714','c_3454']
df_merge_filtré = df_merge[~df_merge['client_id'].isin(client_atypique)]
df_merge_filtré.head()
```

[46]:

	id_prod	price	categ	date	session_id	client_id	\
0	0_0	3.75	0	2021-03-01 13:45:51.575117	s_282	c_5152	
1	0_0	3.75	0	2021-03-02 06:42:55.351333	s_621	c_2917	
2	0_0	3.75	0	2021-03-02 18:49:49.651862	s_852	c_3988	
3	0_0	3.75	0	2021-03-02 21:57:33.862118	s_908	c_1004	
4	0_0	3.75	0	2021-03-03 23:59:48.997483	s_1379	c_278	

	_merge	mois	sex	birth	âge	tranche_age	transaction_id
0	both	2021-03	f	1986	39	36-45 ans	1
1	both	2021-03	m	1988	37	36-45 ans	2
2	both	2021-03	f	1962	63	56-65 ans	3
3	both	2021-03	m	1973	52	46-55 ans	4
4	both	2021-03	f	1987	38	36-45 ans	5

[47]:
```
df_merge_filtré.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 640734 entries, 0 to 687533
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   id_prod         640734 non-null  object
 1   price           640734 non-null  float64
 2   categ           640734 non-null  int64
 3   date            640734 non-null  datetime64[ns]
 4   session_id      640734 non-null  object
 5   client_id       640734 non-null  object
 6   _merge          640734 non-null  category
 7   mois            640734 non-null  period[M]
 8   sex             640734 non-null  object
 9   birth           640734 non-null  int64
 10  âge             640734 non-null  int64
 11  tranche_age     640734 non-null  category
 12  transaction_id  640734 non-null  int64
dtypes: category(2), datetime64[ns](1), float64(1), int64(4), object(4),
period[M](1)
memory usage: 59.9+ MB
```

Le dataset des 3 fichiers fusionnés "df_merge_filtré" sans les 4 clients BtoB va servir de base pour l'analyse des ventes et des corrélations.

1.6 3.3 Evolution du chiffre d'affaires mensuel

[48]:
```
#Conversion des dates en mois
df_merge_filtré.loc[:,'mois'] = df_merge['date'].dt.to_period('M')
```

[49]:
```
#Création d'un dataset des ventes mensuelles
ca_mensuel = df_merge_filtré.groupby(['mois']) [['price']].sum().reset_index()

#Affichage du dataset
ca_mensuel.head()
```

[49]:

	mois	price
0	2021-03	445918.71
1	2021-04	439337.85
2	2021-05	454887.46
3	2021-06	447102.17
4	2021-07	447593.15

[50]:
```
#Trie du dataframe 'CA mensuel' par date pour le calcul de la moyenne mobile
ca_mensuel = ca_mensuel.sort_values(by='mois')

#Calcul de la moyenne mobile sur 3 mois
ca_mensuel['CA_Moyenne_mobile'] = round(ca_mensuel['price'].
 ↪rolling(window=3,min_periods=1).mean(),2)

#Renommer la colonne 'price' par montant du chiffre d'affaires
ca_mensuel = ca_mensuel.rename(columns= {'price':"Chiffre d'affaires"})

#Affichage du dataset
ca_mensuel.head()
```

[50]:

	mois	Chiffre d'affaires	CA_Moyenne_mobile
0	2021-03	445918.71	445918.71
1	2021-04	439337.85	442628.28
2	2021-05	454887.46	446714.67
3	2021-06	447102.17	447109.16
4	2021-07	447593.15	449860.93

[51]:
```
#Visualisation de la moyenne mobile
# Conversion en string
ca_mensuel['mois'] = ca_mensuel['mois'].astype(str)

fig = px.line(
    ca_mensuel,
    x='mois',
    y=["Chiffre d'affaires", 'CA_Moyenne_mobile'],
    labels={'value':"Chiffre d'affaires (€)", 'mois':"Mois", 'variable':
 ↪'Légende'},
```

```
    title= "Evolution du chiffre d'affaires mensuel"
    )
fig.show()
```

Au lancement du site, on a une augmentation des ventes pendant plusieurs mois jusqu'à atteindre un pic en Février 2022. Puis le chiffre d'affaires se stabilise autour de 470000€.

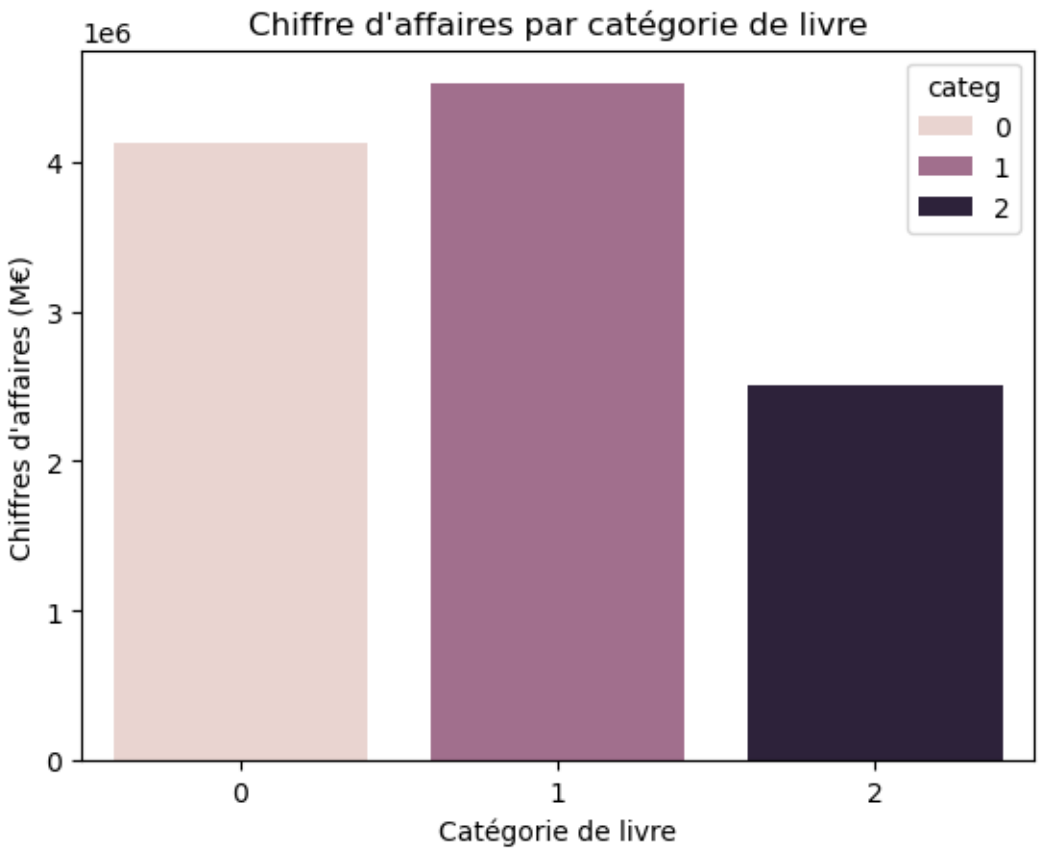
1.7 3.3 Chiffre d'affaires par catégorie

[52]:
```
#Affichage du CA total
CA_total = round(df_merge_filtré['price'].sum(),2)
print(f"Le chiffre d'affaires total est de {CA_total }€ en 2 ans depuis le␣
 ↪lancement du site e-commerce")

#Moyenne de chiffre d'affaires mensuel
moy_ca_mois = round(df_merge_filtré.groupby('mois')['price'].sum().mean(),2)
print(f"Le chiffre d'affaires moyen par mois est de {moy_ca_mois }€ ")
```

```
Le chiffre d'affaires total est de 11143367.01€ en 2 ans depuis le lancement du
site e-commerce
Le chiffre d'affaires moyen par mois est de 464306.96€
```

[53]:
```
#Création du dataframe du CA par catégorie
CA_par_categorie = df_merge_filtré.groupby('categ')['price'].sum().reset_index()
print(CA_par_categorie )

#Visualisation
sns.barplot(data=CA_par_categorie, x=CA_par_categorie['categ'],␣
 ↪y=CA_par_categorie['price'], hue='categ')
plt.title("Chiffre d'affaires par catégorie de livre")
plt.xlabel("Catégorie de livre")
plt.ylabel("Chiffres d'affaires (M€)")
plt.show()
```

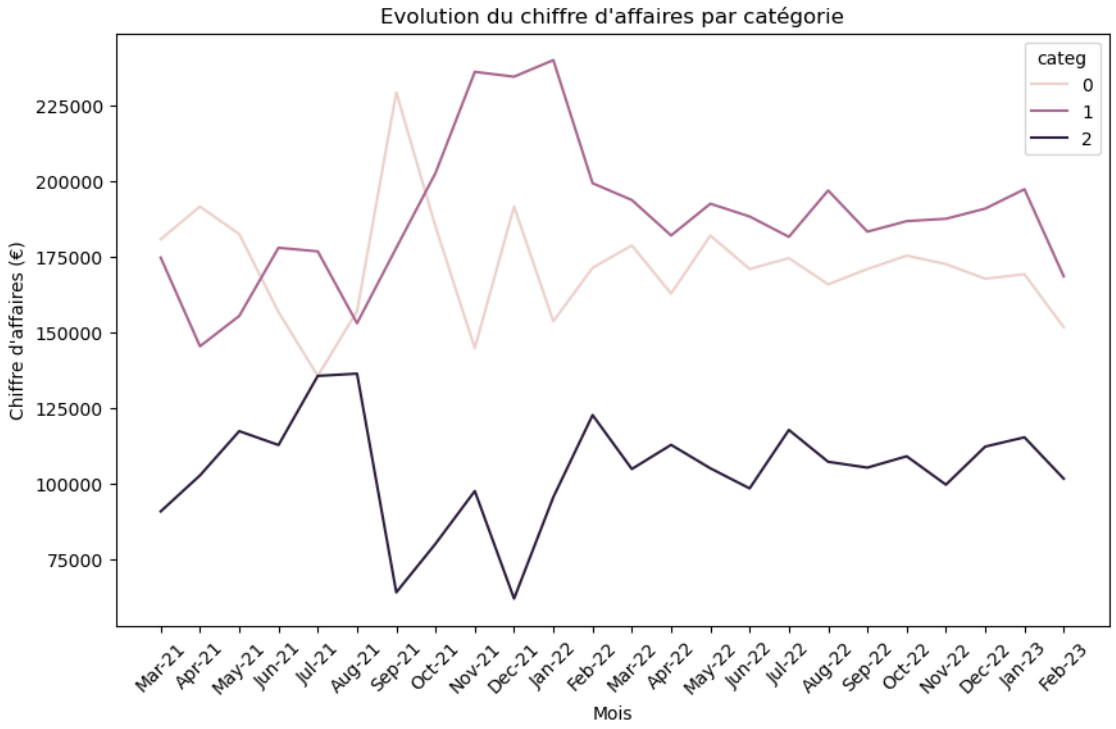
```
   categ       price
0      0  4119200.69
1      1  4520101.86
2      2  2504064.46
```



La catégorie 1 est celle qui génèrent le plus de chiffre d'affaires et celle qui génèrent le moins est la catégorie 2. Cela peut s'expliquer par le fait que les catégories 0 et 1 ont des prix moins onéreux et par conséquent plus de quantités sont vendues.

[54]:
```
#Evolution du Chiffre d'affaires mensuel par catégorie
CA_mois_cat = df_merge_filtré.groupby(['mois', 'categ'])['price'].sum().
 ↪reset_index()

#Conversion des mois au format string (Jan 2025) pour le graphique
CA_mois_cat['mois'] = CA_mois_cat['mois'].dt.strftime('%b-%y')

#Visualisation
plt.figure(figsize=(10,6))
sns.lineplot(data= CA_mois_cat, x='mois',y='price',hue='categ')

plt.title("Evolution du chiffre d'affaires par catégorie")
plt.xticks(rotation=45)
plt.xlabel("Mois")
plt.ylabel("Chiffre d'affaires (€)")
```

[54]: Text(0, 0.5, "Chiffre d'affaires (€)")



Les catégories de livre sont marquées par une saisonnalité de vente:

–> La catégorie 0 : elle génère le plus de CA à la rentrée scolaire et pendant la période de Noël. Le CA chute en période de vacance scolaire.

–> La catégorie 1: hausse de CA à partir de l'été jusqu'en fin d'année et puis ça se stabilise sur le reste des mois

–> La catégorie 2: au contraire, c'est en fin d'année que le CA est le plus bas et le reste de l'année, le CA est stable.

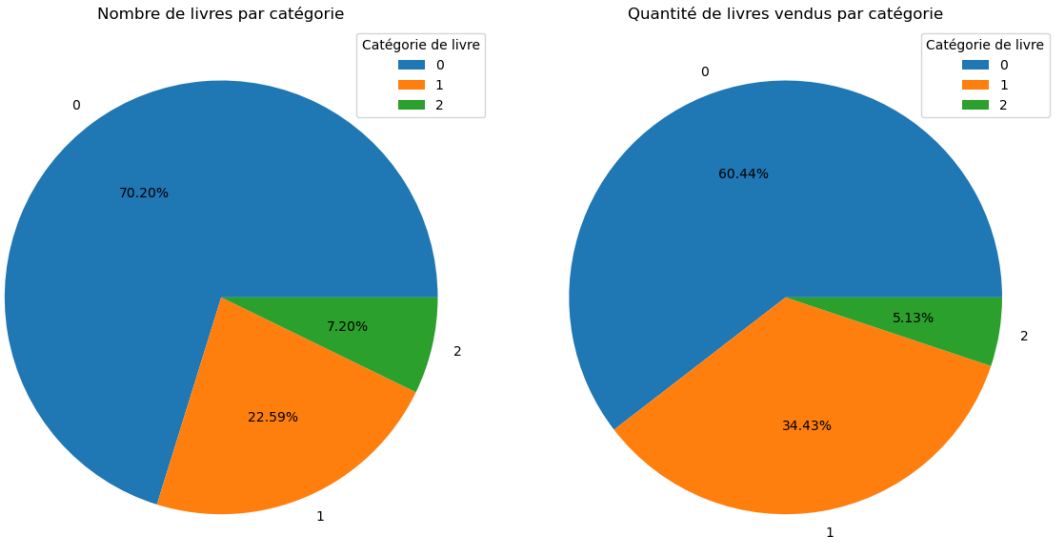
Les catégories 0 et 2 semblent suivent des tendances de hausse et de baisse contraires au même moment. Lorsqu'une catégorie connaît une hausse de CA, l'autre subit une baisse de son CA.

1.8 3.4 Quantité de produits vendus par mois

[55]:
```
#Nombre de livres dans chaque catégorie
Nb_livre_par_categorie = df_merge_filtré.groupby('categ')['id_prod'].nunique().
  ↪reset_index()
print(Nb_livre_par_categorie)

#Quantité de livre vendus pour chaque catégorie de livre
Qte_vendus_par_categorie = df_merge_filtré.groupby('categ')['id_prod'].count().
  ↪reset_index()
```

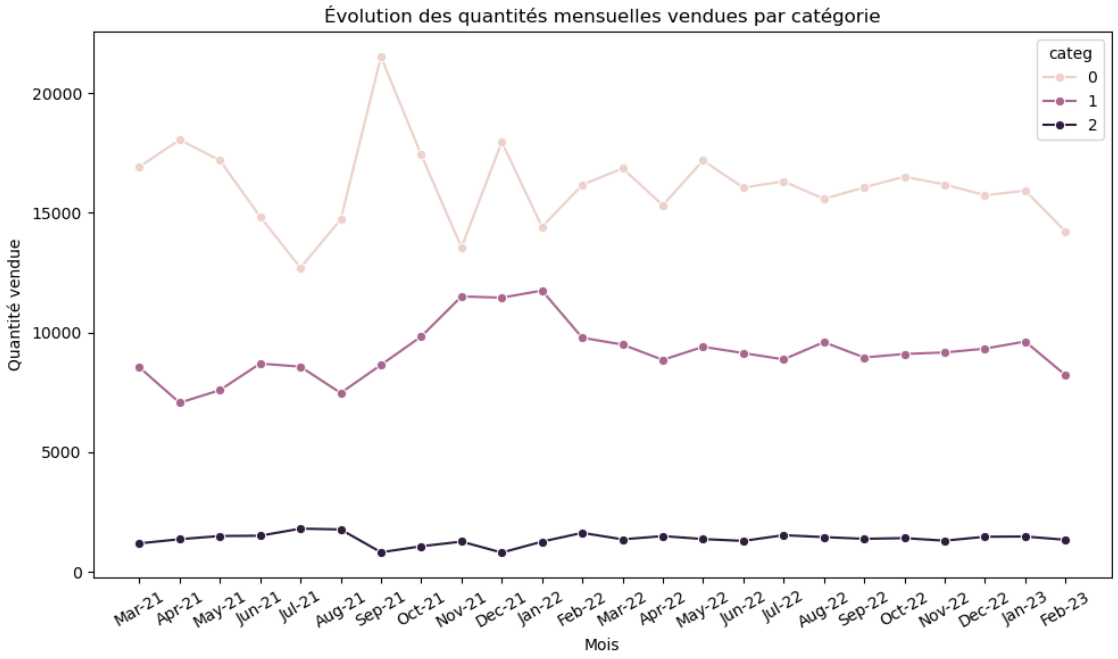
```
print(Qte_vendus_par_categorie)


# Création d'une figure avec 2 subplots horizontalement (1 ligne, 2 colonnes)
fig, axes = plt.subplots(1, 2, figsize=(12, 6))  # axes est un tableau avec deux axes

# 1er graphique dans le subplot de gauche (axes[0])
axes[0].pie(
    Nb_livre_par_categorie['id_prod'],
    labels=Nb_livre_par_categorie['categ'],
    autopct=lambda x: f"{x:.2f}%",
)
axes[0].set_title("Nombre de livres par catégorie")
axes[0].legend(title='Catégorie de livre', loc='best')

# 2ème graphique dans le subplot de droite (axes[1])
axes[1].pie(
    Qte_vendus_par_categorie['id_prod'],
    labels=Qte_vendus_par_categorie['categ'],
    autopct=lambda x: f"{x:.2f}%",
)
axes[1].set_title("Quantité de livres vendus par catégorie")
axes[1].legend(title='Catégorie de livre', loc='best')

plt.tight_layout()
plt.show()
```

```
   categ  id_prod
0      0     2290
1      1      737
2      2      235
   categ  id_prod
0      0   387281
1      1   220605
2      2    32848
```



La catégorie 0 représente 70,2% de l'offre total de livre dans la librairie et elle représente 60.44% des quantités vendues.

[56]:
```
# Compter le nombre de quantité par mois et par catégorie
qte_par_mois = df_merge_filtré.groupby(['categ', 'mois'])['id_prod'].count().
 ↪reset_index().rename(columns={'id_prod': 'quantité'})

# Convertir mois en datetime pour la visualisation
qte_par_mois['mois'] = qte_par_mois['mois'].dt.strftime('%b-%y')

plt.figure(figsize=(10, 6))
sns.lineplot(data=qte_par_mois, x='mois', y='quantité', hue='categ', marker='o')
plt.title("Évolution des quantités mensuelles vendues par catégorie")
plt.xlabel('Mois')
plt.ylabel("Quantité vendue")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```



La catégorie 0 reste celle la plus vendue en terme de volume quelque soit le mois de l'année. Les quantités vendues pour cette catégorie connaît un pic entre août 2021 et octobre 2021 (Période qui coincide avec la rentrée scolaire) et en décembre 2021 (Période de fête de Noël).

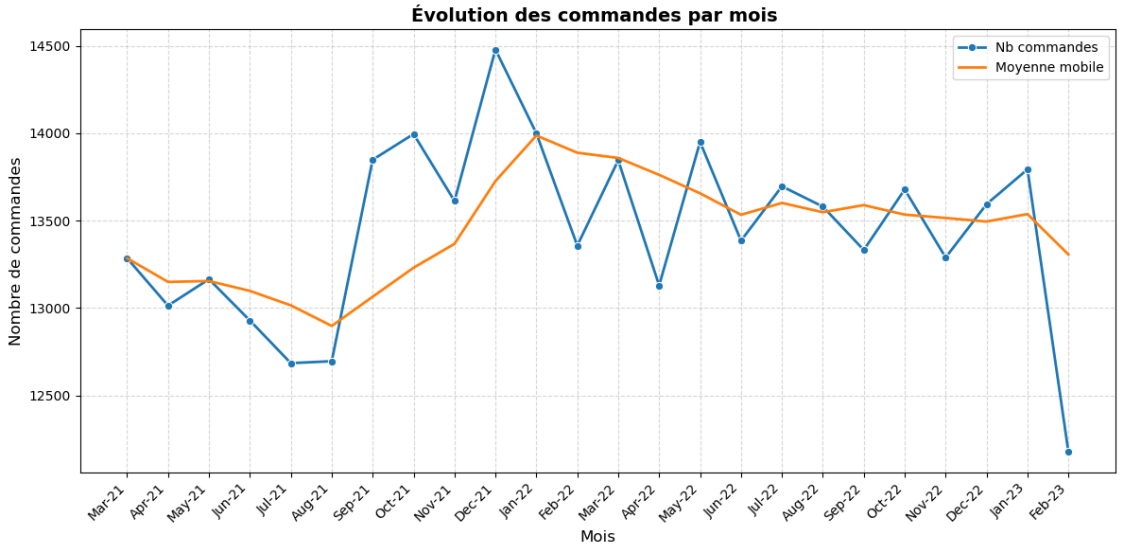
La catégorie 1 se place en 2ème position des catégories les plus vendues en volume. Les quantités augmentent entre août 2021 et novembre 2021 pour se stabiliser en nov 2021 et Jan 2022.

La catégorie 2 se vend le moins en volume et les quantités vendues restent constantes tous les mois de l'année.

Pour la 1ère année de lancement, il y a plus de fluctuations que l'année qui suit où c'est plus stable.

[57]:
```
#Nombre de commandes passées par mois
Commande_mois= df_merge_filtré.groupby(['mois']) ['session_id'].nunique().
 ↪reset_index()

#Conversion de la colonne mois au format string (Mois-Année)
Commande_mois['mois'] = Commande_mois['mois'].dt.strftime('%b-%y')

#Création d'une moyenne mobile du nombre de commande
Commande_mois['moyenne_mobile'] = Commande_mois['session_id'].
 ↪rolling(window=5,min_periods=1).mean()

#Visualisation
plt.figure(figsize=(12, 6))
sns.lineplot(data=Commande_mois, x='mois', y='session_id', marker='o',␣
 ↪linewidth=2, label='Nb commandes')
```

```
sns.lineplot( data=Commande_mois, x='mois', y='moyenne_mobile',linewidth=2,␣
 ↪label= 'Moyenne mobile')

plt.title("Évolution des commandes par mois ", fontsize=14, fontweight='bold')
plt.xlabel('Mois', fontsize=12)
plt.ylabel('Nombre de commandes', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



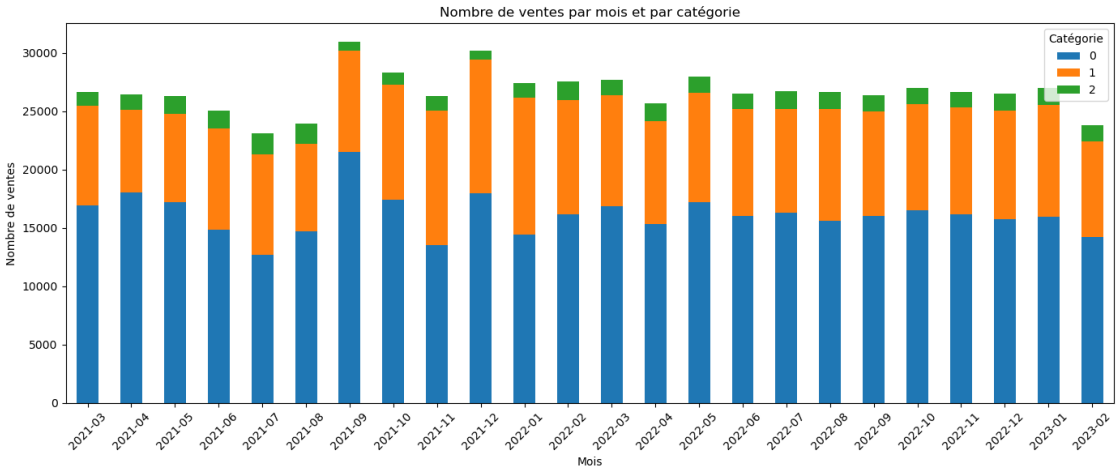
Dans les 1ers mois, le nombre de commandes chutent en période d'été. Les commandes repartent à la hausse à partir de la rentrée scolaire (Sept 2021) jusqu'au nouvel an pour ensuite se stabiliser tout au long de l'année 2022.

1.9 3.5 Nombre de ventes mensuelles

[58]:
```
#Calcul du volume de ventes mensuelles
ventes_par_mois_cat = df_merge_filtré.groupby(['mois', 'categ'])['session_id'].
 ↪count().unstack().fillna(0)

print(f'Moyenne du nombre de ventes mensuelles : {ventes_par_mois_cat.sum().
 ↪mean()}')
#Visualisation
ventes_par_mois_cat.plot(
    kind='bar',
    stacked=True,
    figsize=(14,6))
```

```
plt.title("Nombre de ventes par mois et par catégorie")
plt.xlabel("Mois")
plt.ylabel("Nombre de ventes")
plt.xticks(rotation=45)
plt.legend(title='Catégorie')
plt.tight_layout()
plt.show()
```

```
Moyenne du nombre de ventes mensuelles : 213578.0
```



Chaque mois depuis l'ouverture du site, la catégorie 0 est la plus vendue suivi par la catégorie 1 et en dernier la catégorie 2. Les livres de catégorie 0 et 1 sont les plus présentes dans la librairie et elles ont des prix accessibles. Alors que l'offre de livres de catégorie 2 est la moins importante et plus coûteuse.

[59]:
```
# Nombre total de transactions
Nb_transaction = len(df_merge_filtré)
print("Nombre total de transactions:",Nb_transaction)
```

```
Nombre total de transactions: 640734
```

[60]:
```
#Moyenne d'articles vendus par vente
client_ca_nb_ventes = df_merge_filtré.groupby(['client_id']).agg({'id_prod':
 ↪'count','session_id':'nunique'})

client_ca_nb_ventes['Nb article par commande'] =␣
 ↪(client_ca_nb_ventes['id_prod'] / client_ca_nb_ventes['session_id'])

print(f"Nombre moyen d'articles achetés par vente :␣
 ↪{round(client_ca_nb_ventes['Nb article par commande'].mean(),2)}")
```

```
Nombre moyen d'articles achetés par vente : 1.98
```

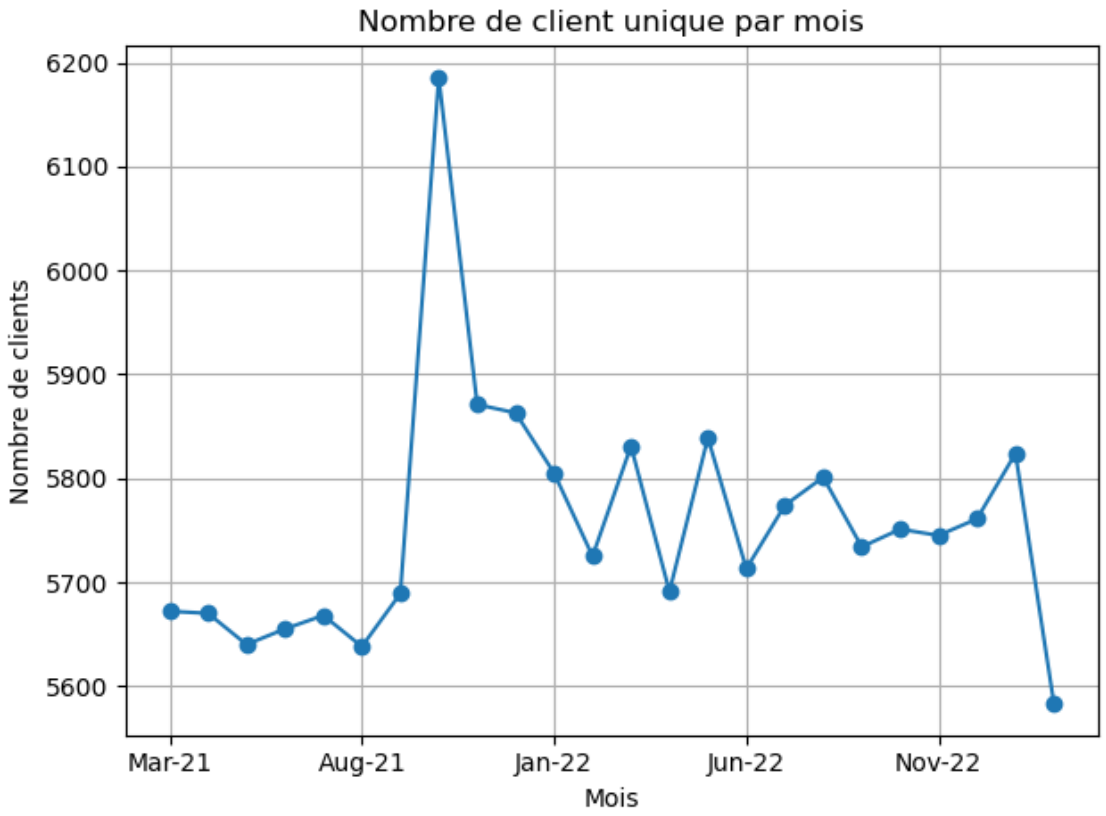
[61]:
```
#Fréquence d'achat par mois par client
freq_achat_mois = df_merge_filtré.groupby(['mois','client_id']) ['session_id'].
 ↪nunique().reset_index()

print(f"Un client passe en moyenne environ {round(freq_achat_mois['session_id'].
 ↪mean(),2)} commandes par mois")
```

```
Un client passe en moyenne environ 2.33 commandes par mois
```

1.10 3.6 Nombre de clients par mois

[62]:
```
#Nombre de clients par mois
nb_clients_mois = df_merge_filtré.groupby('mois') ['client_id'].nunique().
 ↪reset_index(name='nb_clients')
print(f'En moyenne, il y a {round(nb_clients_mois['nb_clients'].mean(),1)}␣
 ↪clients par mois')

#Conversion des mois au format string mois-année (Jan 2025)
nb_clients_mois['mois'] = nb_clients_mois['mois'].dt.strftime('%b-%y')

#Visualisation
nb_clients_mois.plot(x='mois',
                     y='nb_clients',
                     kind='line',
                     marker='o',
                     title="Nombre de client unique par mois",
                     legend=False)
plt.xlabel("Mois")
plt.ylabel("Nombre de clients")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
En moyenne, il y a 5755.4 clients par mois
```



On remarque un pic du nombre de clients qui ont passé une commande au mois d'octobre 2021. Cela peut-être dû à la rentrée scolaire qui se fait en septembre où la demande en livre scolaire est la plus conséquente. Ou bien une opération marketing qui a attiré l'attention des clients. Par contre, on a pas connu le même pic l'année suivante.

[63]:
```
#Création de la colonne 'jour'
df_merge_filtré.loc[:,'jour'] = df_merge_filtré['date'].dt.to_period('D')

#Création de la colonne pour avoir le nom du jour
df_merge_filtré.loc[:,'jour_nom'] = df_merge_filtré['date'].dt.day_name()
```

```
C:\Users\mende\AppData\Local\Temp\ipykernel_5768\1010056214.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

C:\Users\mende\AppData\Local\Temp\ipykernel_5768\1010056214.py:5:
```

```
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
```

[64]:
```
#Identifier le top 10 des clients qui ont passé le plus de commande en Octobre␣
 ↪2021
octobre = df_merge_filtré[df_merge_filtré['mois'] == '2021-10'].
 ↪sort_values(by='jour',ascending=True)
octobre.groupby('client_id',observed=True).agg({'session_id':
 ↪'nunique','tranche_age':'first','price':'sum','categ':'first'}).
 ↪rename(columns={'session_id':'Nb de commande','price':'CA octobre 2021'}).
 ↪sort_values(by= 'Nb de commande',ascending=False).head(10)
```

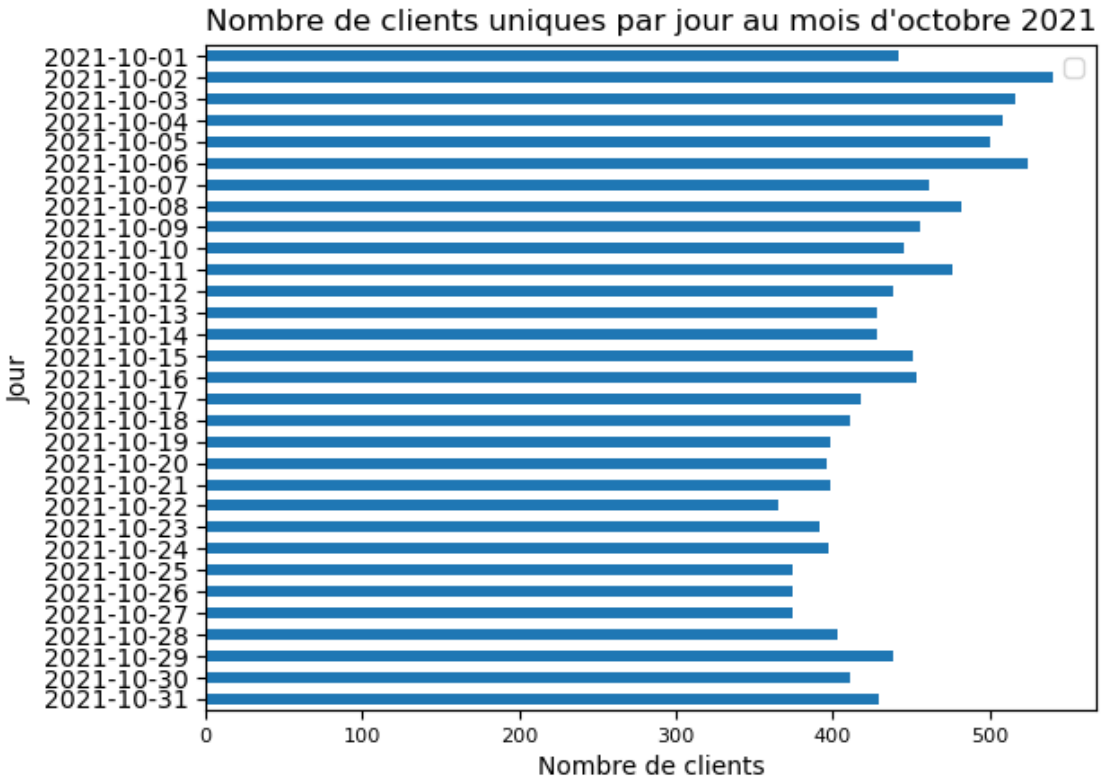
[64]:

client_id	Nb de commande	tranche_age	CA octobre 2021	categ
c_8556	15	46-55 ans	513.40	1
c_1734	13	36-45 ans	376.44	0
c_682	13	46-55 ans	423.60	0
c_7959	13	46-55 ans	438.46	0
c_81	12	46-55 ans	345.36	0
c_1343	11	26-35 ans	235.13	1
c_4871	11	36-45 ans	285.06	1
c_1637	11	36-45 ans	329.86	1
c_7262	10	46-55 ans	396.65	0
c_1368	10	36-45 ans	256.20	0

[65]:
```
#Visualisation du nombre de clients par jour sur tous le mois d'octobre 2021
plt.figure(figsize=(10,6))
octobre.groupby(['jour','jour_nom']) ['client_id'].nunique().reset_index().
 ↪sort_values(by='jour',ascending=False).plot(
    kind='barh',
    x='jour',
    y='client_id'
)
plt.title("Nombre de clients uniques par jour au mois d'octobre 2021")
plt.xticks(rotation=90,fontsize=8)
plt.ylabel('Jour')
plt.xticks(rotation=0)
plt.xlabel("Nombre de clients")
plt.legend(labels='')
```

[65]: <matplotlib.legend.Legend at 0x22b0eea34d0>

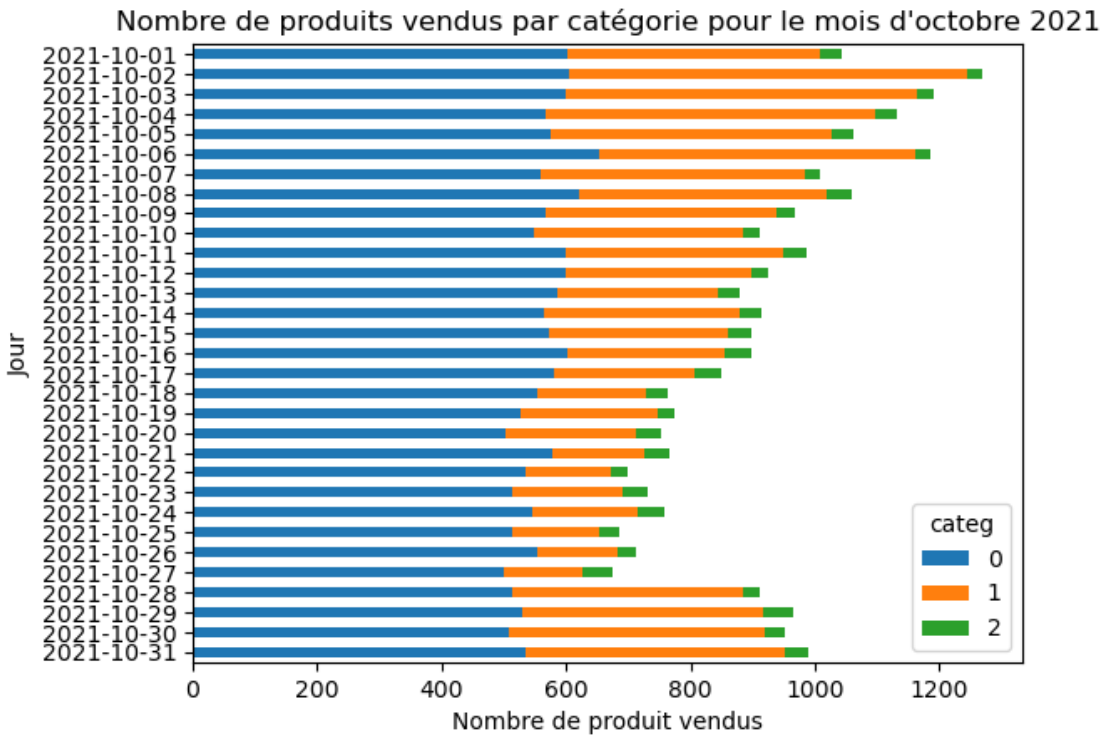
<Figure size 1000x600 with 0 Axes>



C'est le samedi 2 octobre 2021 où l'on enregistre près de 600 clients uniques qui ont passé commande sur le site. C'est le nombre le plus important de clients sur tous le mois. C'est les 1ères semaines de ce mois où il y a le plus de clients.

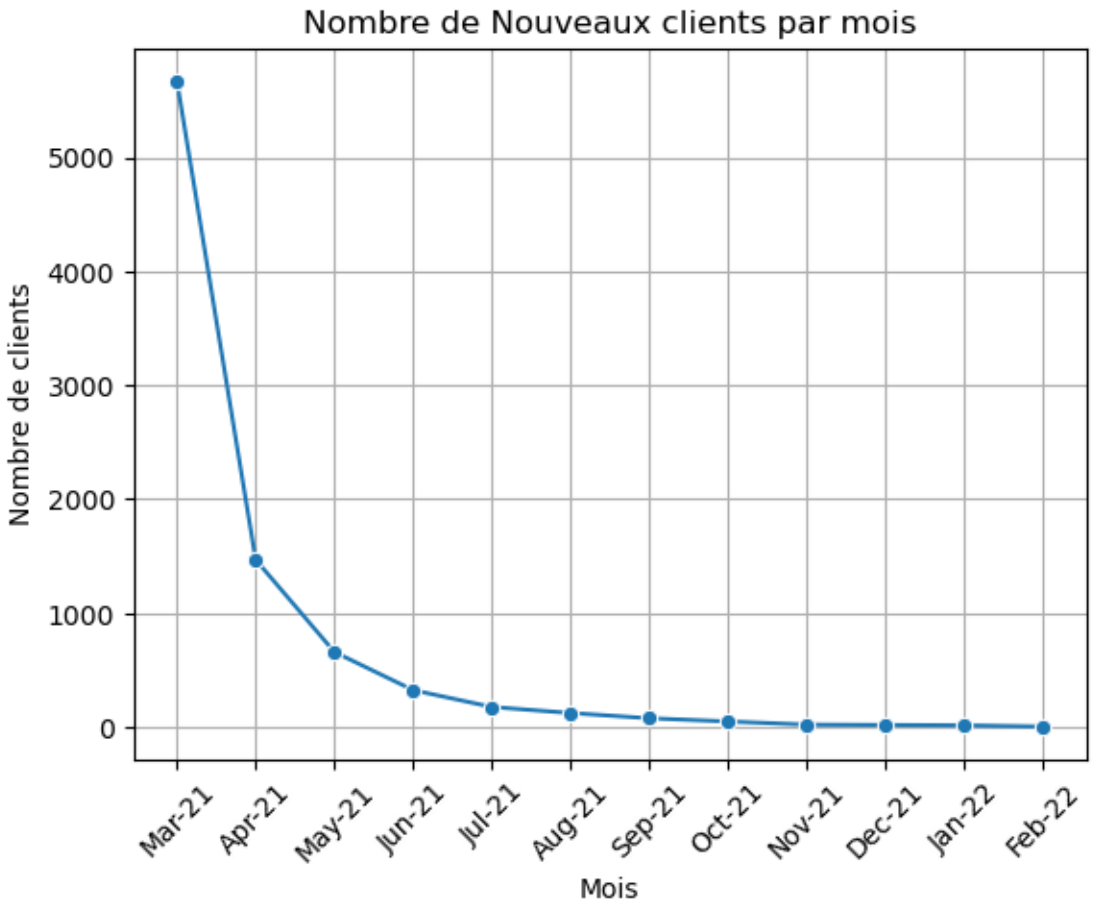
[66]:
```
#Visualisation du nombre de produit vendus par catégorie pour chaque jour du
 ↪mois d'octobre
octobre.groupby(['jour','categ']) ['id_prod'].count().unstack().
 ↪sort_values(by='jour',ascending=False).plot(
    kind='barh',
    stacked=True
)
plt.title("Nombre de produits vendus par catégorie pour le mois d'octobre 2021")
plt.xlabel("Nombre de produit vendus")
plt.ylabel("Jour")
```

[66]: Text(0, 0.5, 'Jour')



[67]:
```
#Récuperer le mois de la 1ère commande pour chaque client
premier_achat = df_merge_filtré.groupby(['client_id']) ['mois'].min().
 ↪reset_index()

#Compter le nombre de nouveaux clients par mois
Nb_nvx_client_mois = premier_achat.groupby(['mois']) ['client_id'].nunique().
 ↪reset_index()

#Afficher les mois sous format string Mois-Année
Nb_nvx_client_mois['mois'] = Nb_nvx_client_mois['mois'].dt.strftime('%b-%y')

#Visualisation
sns.lineplot(
    data = Nb_nvx_client_mois,
    x= 'mois',
    y= 'client_id',
    marker= 'o')

plt.title("Nombre de Nouveaux clients par mois")
plt.xlabel ("Mois")
plt.xticks (rotation= 45)
plt.ylabel ("Nombre de clients")
plt.grid(True)
```

```
plt.show()
```



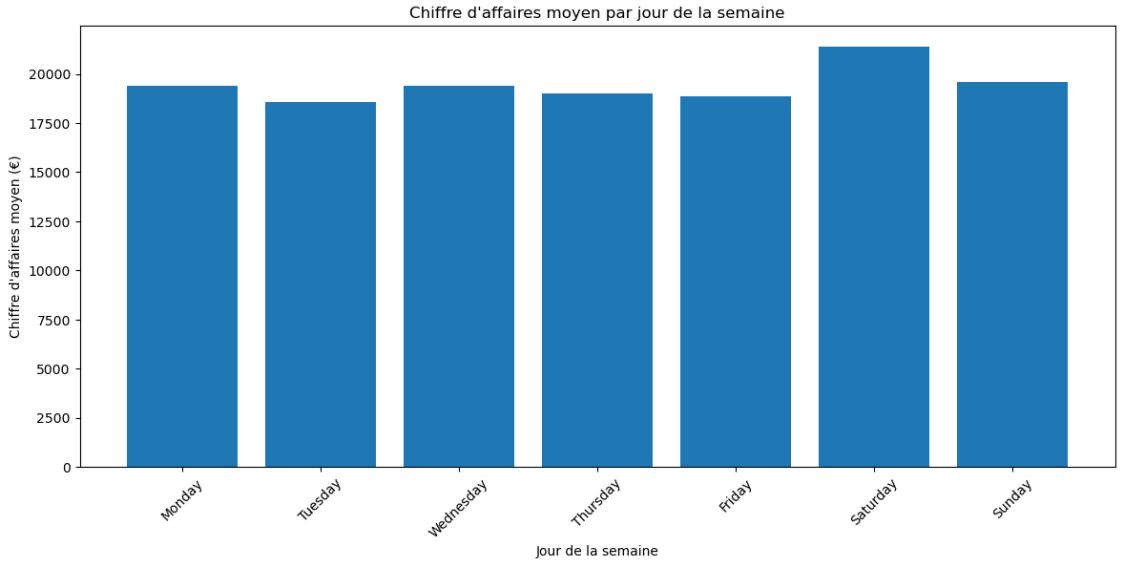
Plus de la moitié des clients ont passé leur 1ère commande au 1er mois d'ouverture du site. Il n' y a pas eu de nouveaux clients depuis Février 2022. Donc l'opération de communication sur le lancement du site a plutôt bien fonctionné auprès de la clientèle qui avait pour habitude de se rendre en librairie.

1.11 3.7 Chiffre d'affaires journalier

[68]:
```
# Extraction du jour et du nom du jour
df_merge_filtré.loc[:,'jour'] = df_merge_filtré['date'].dt.date
df_merge_filtré.loc[:,'jour_nom'] = df_merge_filtré['date'].dt.day_name()

# CA par jour (date)
CA_par_jour = df_merge_filtré.groupby(['jour','jour_nom'])['price'].sum().
 ↪reset_index(name='CA')

# Moyenne du CA par nom du jour de la semaine
```

```
CA_moyen_par_jour = CA_par_jour.groupby(['jour','jour_nom'])['CA'].mean().
 ↪reset_index()

# Ordonner les jours de la semaine
ordre_jour =␣
 ↪['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
CA_moyen_par_jour.loc[:,'jour_nom'] = pd.
 ↪Categorical(CA_moyen_par_jour['jour_nom'], categories=ordre_jour,␣
 ↪ordered=True)

# Visualisation
plt.figure(figsize=(14, 6))
plt.bar(CA_moyen_par_jour['jour_nom'], CA_moyen_par_jour['CA'])
plt.title("Chiffre d'affaires moyen par jour de la semaine")
plt.xlabel("Jour de la semaine")
plt.ylabel("Chiffre d'affaires moyen (€)")
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\mende\AppData\Local\Temp\ipykernel_5768\1380658124.py:2: FutureWarning:

Setting an item of incompatible dtype is deprecated and will raise in a future
error of pandas. Value '[datetime.date(2021, 3, 1) datetime.date(2021, 3, 2)
 datetime.date(2021, 3, 2) … datetime.date(2022, 11, 11)
 datetime.date(2022, 12, 11) datetime.date(2023, 1, 21)]' has dtype incompatible
with period[D], please explicitly cast to a compatible dtype first.
```



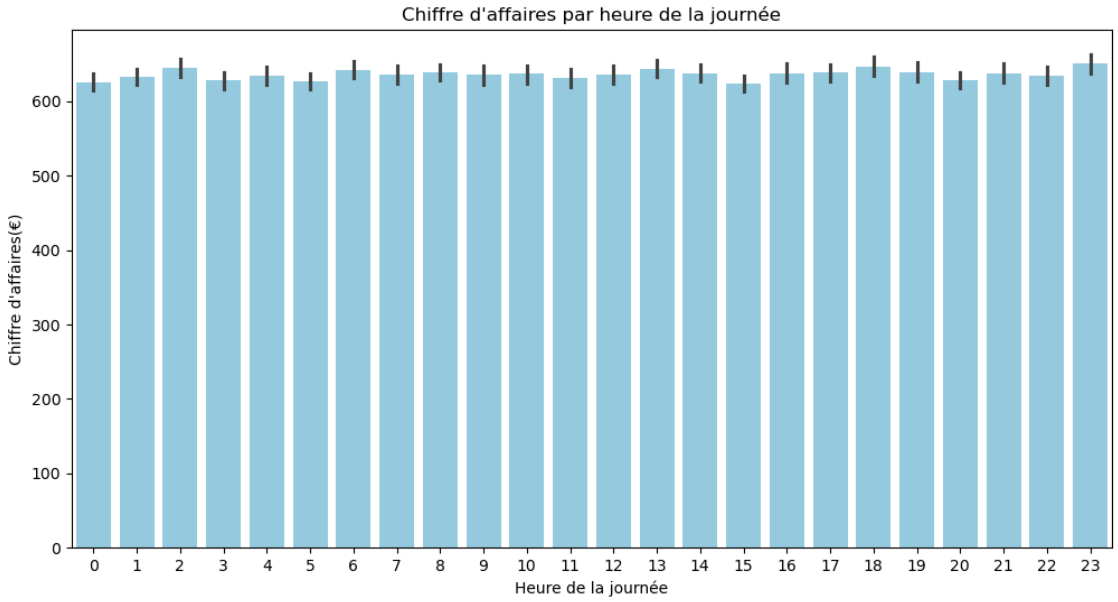
On a eu à peu près le même chiffres d'affaires à part le samedi où il y a un peu plus de chiffre

d'affaires.

[69]:
```
#Nombre de commandes par jour
Nb_cde_jour = df_merge_filtré.groupby(['jour', 'jour_nom']) ['session_id'].
 ↪nunique().reset_index()

#Moyenne de commande par jour
moyenne_commandes_par_jour =  Nb_cde_jour['session_id'].mean()
print(f'Moyenne de commandes par jour : {moyenne_commandes_par_jour}')

#Nombre de commande par heure
df_merge_filtré.loc[:,'heure'] = pd.to_datetime(df_merge_filtré['date']).dt.hour
Nb_cde_heure = df_merge_filtré.groupby(['heure','jour', 'jour_nom'])␣
 ↪['session_id'].nunique().reset_index(name='nb_commandes')

#Moyenne commandes par heure
moyenne_cde_heure = Nb_cde_heure['nb_commandes'].mean()
print(f'Moyenne de commandes par heure : {moyenne_cde_heure}')

#CA par heure
df_merge_filtré.loc[:,'heure'] = pd.to_datetime(df_merge_filtré['date']).dt.hour
ca_heure = df_merge_filtré.groupby(['heure','jour', 'jour_nom']) ['price'].
 ↪sum().reset_index(name='CA')
print(f"Moyenne du Chiffre d'affaires par heure: {round(ca_heure['CA'].
 ↪mean(),2)}€")
```

```
Moyenne de commandes par jour : 443.7671232876712

C:\Users\mende\AppData\Local\Temp\ipykernel_5768\1715431415.py:9:
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


Moyenne de commandes par heure : 22.45733203949997
Moyenne du Chiffre d'affaires par heure: 636.07€
```

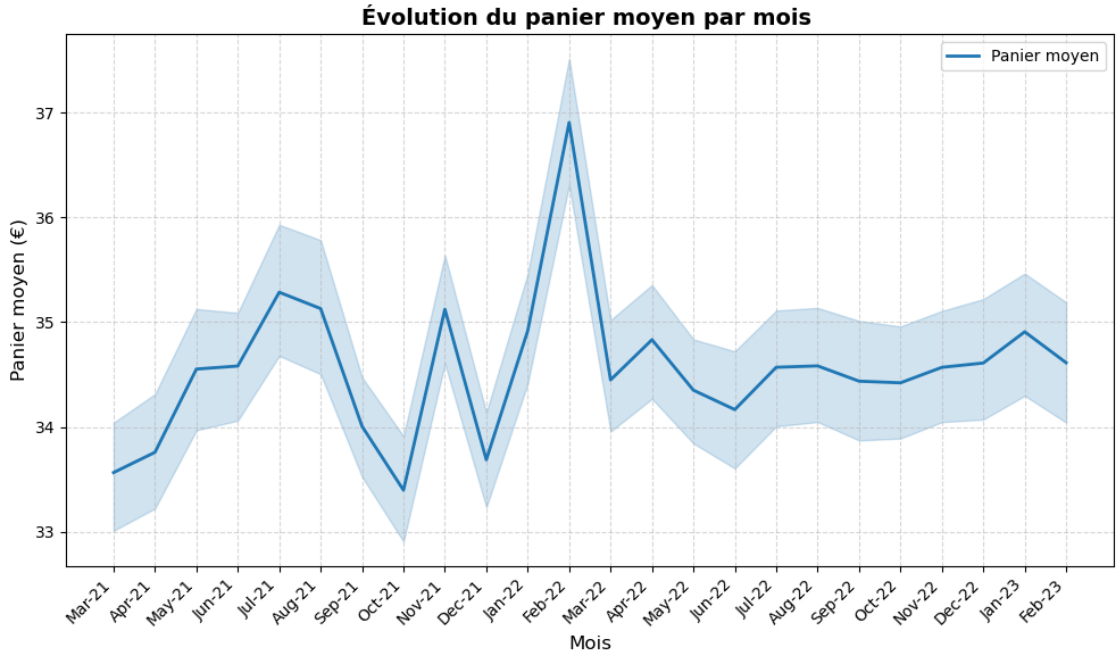
[70]:
```
#Visualisation
plt.figure(figsize=(12,6))
sns.barplot(data=ca_heure, x='heure', y='CA', color='skyblue')
plt.title("Chiffre d'affaires par heure de la journée")
plt.xlabel("Heure de la journée")
plt.ylabel("Chiffre d'affaires(€)")
plt.show()
```



1.12 3.8 Panier moyen par client

[71]:
```
# Calcul du panier moyen par commande
montant_par_commande = df_merge_filtré.groupby(['session_id','mois']).
 ↪agg({'price':'sum'}).reset_index().sort_values(by='mois',ascending=True)

panier_moyen = montant_par_commande['price'].mean() #Moyenne des montants par␣
 ↪commande

print(f"Le panier moyen par commande est de {round(panier_moyen,2)}€")

montant_par_commande['mois'] = montant_par_commande['mois'].dt.
 ↪strftime('%b-%y') #Modification du format pour le graphique

#Visualisation
plt.figure(figsize=(10,6))
sns.lineplot(data= montant_par_commande, x='mois', y='price', linewidth=2,␣
 ↪label='Panier moyen')

plt.title("Évolution du panier moyen par mois ", fontsize=14, fontweight='bold')
plt.xlabel('Mois', fontsize=12)
plt.ylabel('Panier moyen (€)', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

```
Le panier moyen par commande est de 34.55€
```



Le panier moyen par commande oscille entre 34 et 35€. Mais on observe une première chute en octobre 2021, même mois où l'on a enregistré le plus de visites. Donc il se pourrait que même si il y a eu beaucoup de clients, il n'ont pas forcément dépenser plus que d'habitude. Soldes ? Opération marketing ?

Le mois de Février 2022 est celui où le panier moyen est le plus élevé. C'est le mois où l'on enregistre le plus gros CA.

[72]:
```
ventes_février_2022 = df_merge_filtré[df_merge_filtré['mois'] == "2022-02"]
print(f"Nombre de ventes : {ventes_février_2022.shape[0]}")
print(f"Chiffres d'affaires: {ventes_février_2022['price'].sum()}€")

print(f"Répartition du CA par catégorie: {ventes_février_2022.groupby('categ')␣
↪['price'].sum()}")
```

```
Nombre de ventes : 27569
Chiffres d'affaires: 492927.13€
Répartition du CA par catégorie: categ
0    171210.18
1    199168.20
2    122548.75
Name: price, dtype: float64
```

[73]:
```
#Création du dataset du panier moyen par âge
```

```
age_panier_moyen =  df_merge_filtré.groupby(['client_id','âge','mois']).
 ↪agg({'price':'sum','session_id':'nunique','tranche_age':'first'}).
 ↪reset_index()
age_panier_moyen['panier_moyen'] = round(age_panier_moyen['price'] /␣
 ↪age_panier_moyen['session_id'],2)
age_panier_moyen.head()
```

[73]:

	client_id	âge	mois	price	session_id	tranche_age	panier_moyen
0	c_1	70	2021-06	19.53	1	66-80 ans	19.53
1	c_1	70	2021-07	55.22	4	66-80 ans	13.80
2	c_1	70	2021-08	13.96	1	66-80 ans	13.96
3	c_1	70	2021-09	19.98	2	66-80 ans	9.99
4	c_1	70	2021-10	78.83	2	66-80 ans	39.42

[74]:
```
#Création du dataset du panier moyen par tranche d'âge
tranche_age_panier_moyen = age_panier_moyen.groupby(['tranche_age'],␣
 ↪observed=True) ['panier_moyen'].mean().reset_index()

#Visualisation

tranche_age_panier_moyen = px.bar(
    tranche_age_panier_moyen,
    x='tranche_age', y='panier_moyen',
    title="Panier moyen par tranche d'âge",
    labels={'tranche_age':"Tranche d'âge", 'panier_moyen':"Panier moyen (€)"}
)
tranche_age_panier_moyen.show()
```

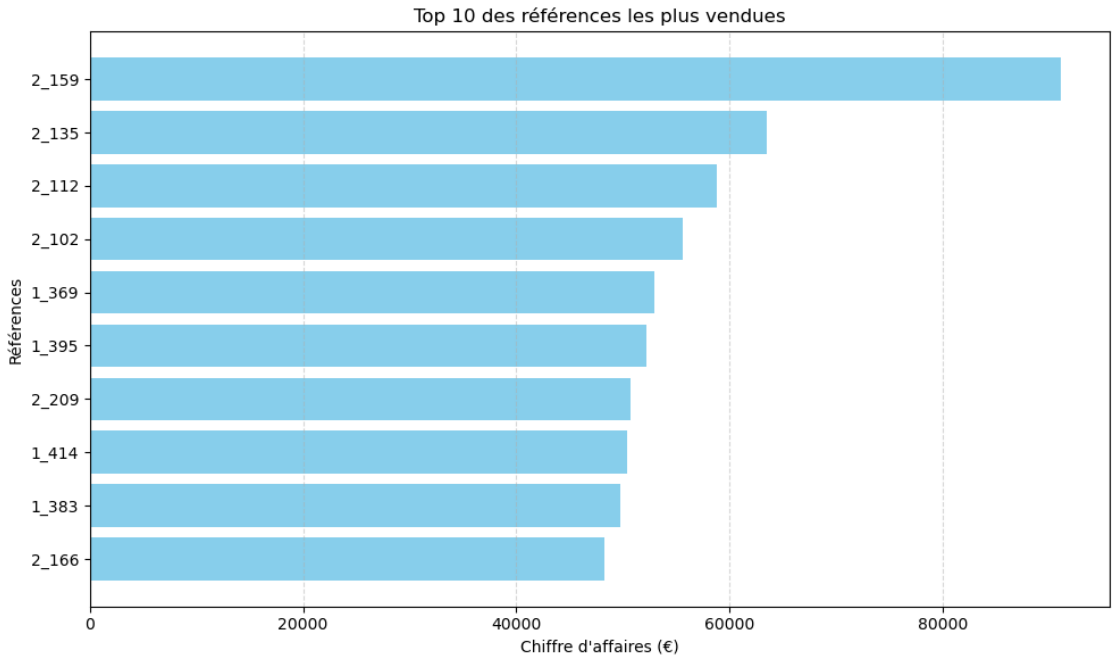
Les clients de 18 à 25 ans (les moins nombreux) et de 26 à 35 ans sont ceux qui dépensent le plus à chaque commande passée. Ils dépensent plus de 60€ par commande.

Les clients de 36 à 45 ans dépensent le moins avec 13€ par commande. C'est une des tranche d'âge la plus représenté au sein de la clientèle.

1.13 3.9 Top 10 des meilleures ventes

[75]:
```
#Top 10 des références les mieux vendus
Top_10_réf = df_merge_filtré.groupby('id_prod') ['price'].sum().
 ↪sort_values(ascending=False).head(10).reset_index()

#Visualisation
plt.figure(figsize=(10,6))
plt.barh(
    y=Top_10_réf['id_prod'],
    width=Top_10_réf['price'],
    color='skyblue'
)
```

```
plt.title("Top 10 des références les plus vendues")
plt.xlabel("Chiffre d'affaires (€)")
plt.ylabel("Références")
plt.gca().invert_yaxis()  # Pour avoir le plus grand en haut
plt.grid(True, axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



[76]:
```
#Liste du top 10
liste_top = Top_10_réf['id_prod']
liste_top
#Filtrer le dataframe sur les 10 meilleures références
Top_10 = df_merge_filtré[df_merge_filtré['id_prod'].isin(liste_top)]
# Regrouper les 10 meilleurs refs par catégorie pour voir la répartition du CA
repartition_categ = Top_10.groupby('categ')['price'].sum().
 ↪reset_index(name='CA')
repartition_categ
```

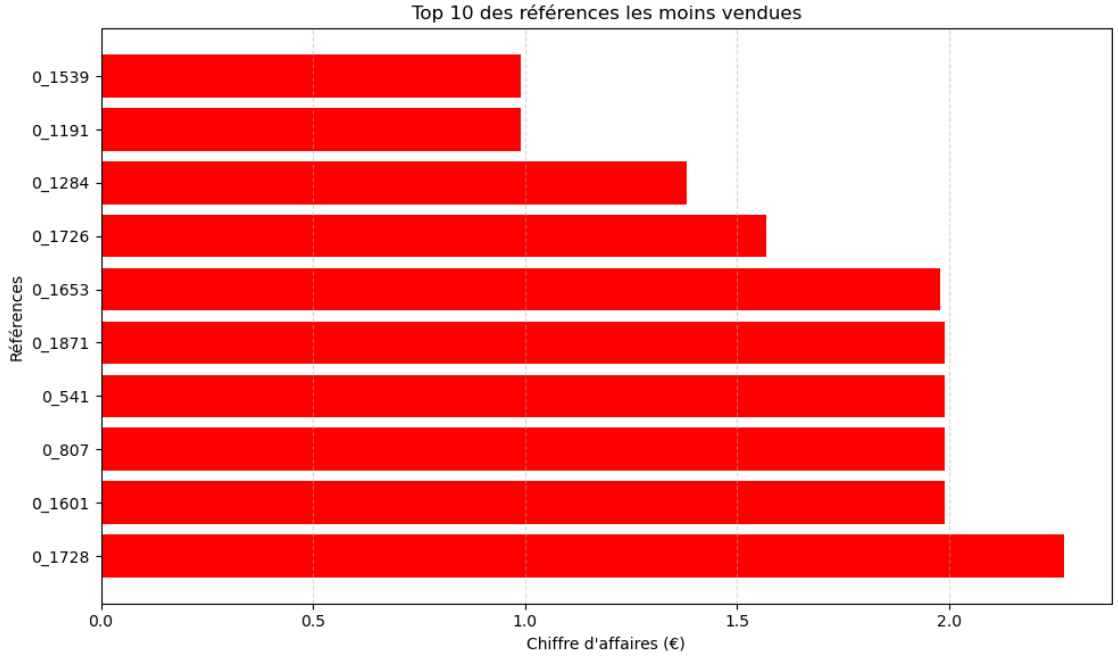
[76]:

	categ	CA
0	1	205203.41
1	2	368056.35

Les 10 références générant le plus de chiffre d'affaires font partie de la catégorie 2 et 1 où les prix de ventes sont plus élevés surtout pour la catégorie 2.

1.14 3.10 Top 10 des pires ventes

[77]:
```
#Top 10 des références les moins vendus
Flop_10_réf = df_merge_filtré.groupby('id_prod') ['price'].sum().
 ↪sort_values(ascending=True).head(10).reset_index()

#Visualisation
plt.figure(figsize=(10,6))
plt.barh(
    y= Flop_10_réf['id_prod'],
    width= Flop_10_réf['price'],
    color='red'
)

plt.title("Top 10 des références les moins vendues")
plt.xlabel("Chiffre d'affaires (€)")
plt.ylabel("Références")
plt.gca().invert_yaxis()  # Pour avoir le plus grand en haut
plt.grid(True, axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



[78]:
```
#Liste du Flop 10
liste_flop = Flop_10_réf['id_prod']
```

```
#Filtrer le dataframe sur les 10 meilleures références
Flop_10 = df_merge_filtré[df_merge_filtré['id_prod'].isin(liste_flop)]

# Regrouper les 10 meilleurs refs par catégorie pour voir la répartition du CA
repartition_categ = Flop_10.groupby('categ')['price'].sum().
 ↪reset_index(name='CA')
repartition_categ
```

[78]:

	categ	CA
0	0	17.14

Les 10 références générant le moins de chiffre d'affaires font partie de la catégorie 0. C'est une catégorie où les prix de ventes sont bas.

[79]:
```
#Nombre de transactions par client
Nb_transaction_client = df_merge_filtré.groupby('client_id') [['session_id']].
 ↪nunique().sort_values(by='session_id', ascending=False).reset_index()
Nb_transaction_client.head()
```

[79]:

	client_id	session_id
0	c_8526	167
1	c_2265	165
2	c_1637	165
3	c_669	163
4	c_8510	162

[80]:
```
#Identifier le top 5 des clients qui ont commandé le plus sur le site
client_top_commande = ['c_1609','c_6714','c_3454','c_4958','c_2140']
clients[clients['client_id'].isin(client_top_commande)]
```

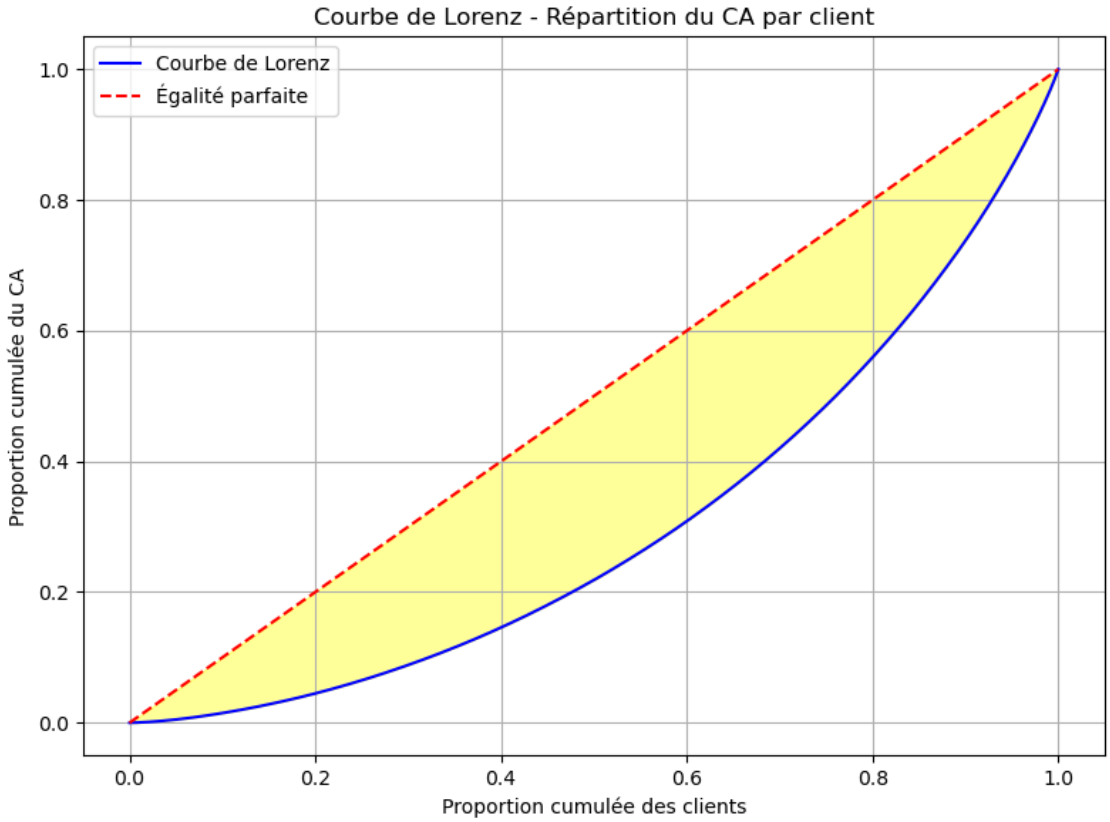
[80]:

	client_id	sex	birth	âge	tranche_age
612	c_2140	f	1977	48	46-55 ans
1378	c_4958	m	1999	26	26-35 ans
1911	c_6714	f	1968	57	56-65 ans
3641	c_1609	m	1980	45	36-45 ans
8087	c_3454	m	1969	56	56-65 ans

1.15 3.11 Top clients par CA

[81]:
```
# 1. Calcul du  CA par client
CA_par_client = df_merge_filtré.groupby('client_id') [['price']].sum().
 ↪reset_index()

# 2. Renommer la colonne 'price' par 'CA'
CA_par_client.rename(columns={'price':'CA'}, inplace=True)


# 3. Trier du plus petit CA au plus grand
```

```
CA_client_tri = CA_par_client.sort_values(by='CA',ascending= True)

# 4. Calcul du cumul du CA  et du cumul client

cumul_client = np.cumsum(np.ones(len(CA_client_tri))) / len(CA_client_tri)
cumul_ca = np.cumsum(CA_client_tri['CA']) / CA_client_tri['CA'].sum()

# 5. Visualisation Courbe Lorenz
plt.figure(figsize=(8,6))
plt.plot(cumul_client, cumul_ca, label="Courbe de Lorenz", color='blue')

 #Création de la diagonale de l'égalité parfaite
plt.plot([0,1], [0,1], '--', color='red', label="Égalité parfaite")
plt.fill_between(cumul_client, cumul_ca, cumul_client, color='yellow', alpha=0.
 ↪4)

plt.title("Courbe de Lorenz - Répartition du CA par client")
plt.xlabel("Proportion cumulée des clients")
plt.ylabel("Proportion cumulée du CA")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



[82]:
```
# Calcul de l'indice de Gini
gini = 1 - 2 * np.trapezoid(cumul_ca,cumul_client)
print(f"Indice de Gini : {round(gini,3)}")
```

```
Indice de Gini : 0.398
```

- L'indice de Gini mesure l'écart entre la diagonale et la courbe.
- Indice de gini proche de 1 –> fortement inégalitaire*
- Indice de gini proche de 0 –> égalitaire*

L'inégalité est modérée

40 % des clients génèrent environ que 15 % du CA.

80 % des clients génèrent environ 55 % du CA.

Donc les 20 % restants (les plus gros clients) génèrent environ 45 % du CA.

[83]:
```
#Tri du CA décroissant pour le cumul du CA dans l'ordre du client le plus␣
 ↪rentable
CA_décroissant = CA_par_client.sort_values(by= 'CA', ascending= False)

#Création de la colonne du CA cumulé du CA en pourcentage
CA_décroissant['% CA cumulé'] = (CA_décroissant['CA'].cumsum() /␣
 ↪CA_décroissant['CA'].sum())*100

#Création de la colonne des clients cumulés en pourcentage
CA_décroissant['% Client cumulé'] = ((CA_décroissant.index + 1)/␣
 ↪len(CA_décroissant))*100
```

[84]:
```
#Identifier les clients qui génèrent 80 % du CA
clients_pareto = CA_décroissant[CA_décroissant['% CA cumulé'] <= 80]
print(f"{len(clients_pareto)} clients sur {len(CA_décroissant)} (soit␣
 ↪{len(clients_pareto)/len(CA_décroissant)*100:.2f}%) génèrent 80% du CA.")
clients_pareto
```

```
4499 clients sur 8596 (soit 52.34%) génèrent 80% du CA.
```

[84]:

	client_id	CA	% CA cumulé	% Client cumulé
634	c_1570	5285.82	0.047435	7.387157
2512	c_3263	5276.87	0.094789	29.234528
1267	c_2140	5260.18	0.141994	14.751047
2107	c_2899	5214.05	0.188784	24.523034
7002	c_7319	5155.77	0.235052	81.468125
…	…	…	…	…
5210	c_5701	994.23	79.959866	60.621219
2612	c_3353	994.06	79.968787	30.397859

```
8567     c_973   993.49    79.977702        99.674267
4724    c_5264   993.46    79.986617        54.967427
8166    c_8369   992.90    79.995528        95.009307

[4499 rows x 4 columns]
```

[85]:
```
#Nombre de clients dans les 20% les plus rentables
print(f"Nombre de clients dans les 20% les plus rentables :␣
 ↪{len(CA_décroissant[CA_décroissant['% CA cumulé'] <=45])}")
```

```
Nombre de clients dans les 20% les plus rentables : 1769
```

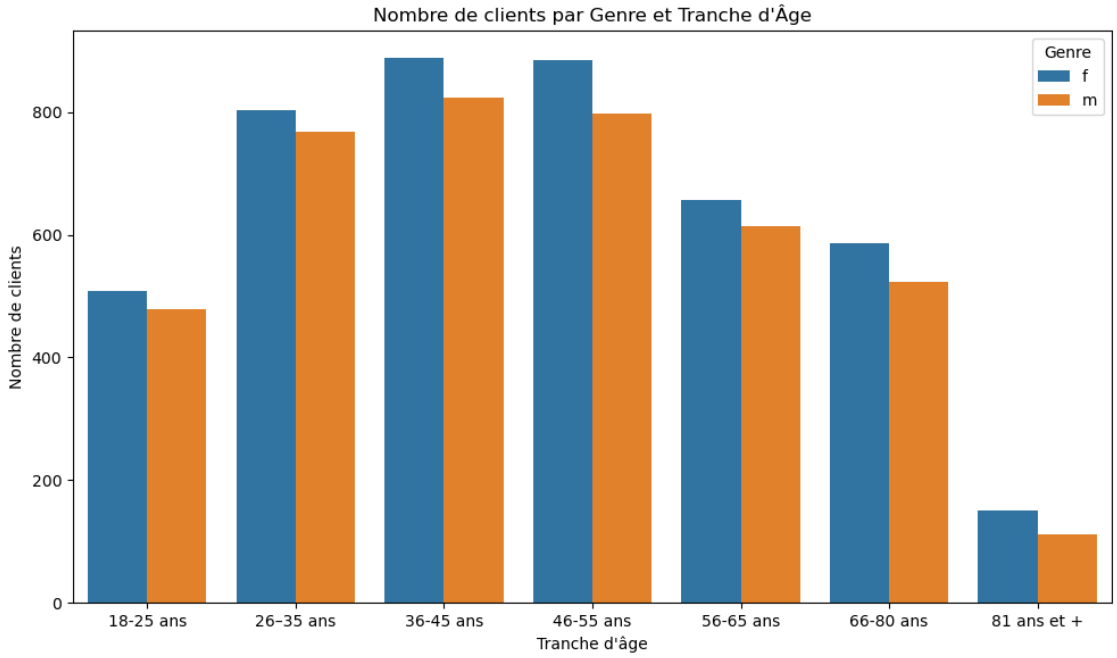
[86]:
```
#Liste du top 20 clients les plus rentables
print(f"top 20 clients les plus rentables :␣
 ↪{CA_décroissant[['client_id','CA',]].head(20)}")
```

```
top 20 clients les plus rentables :      client_id       CA
634     c_1570  5285.82
2512    c_3263  5276.87
1267    c_2140  5260.18
2107    c_2899  5214.05
7002    c_7319  5155.77
7711    c_7959  5135.75
470     c_1422  5131.36
7116    c_7421  5097.18
7787    c_8026  5082.58
1672    c_2505  5059.35
972     c_1876  5026.60
3868    c_4491  5025.05
4723    c_5263  5006.85
1771    c_2595  4959.66
8191    c_8392  4934.01
8465     c_880  4897.19
8323    c_8510  4888.16
706     c_1636  4883.56
1196    c_2077  4848.06
4256    c_4840  4826.51
```

1.16 3.12 Effectif client par tranche d'âge et par genre

[87]:
```
#Effectif client par genre et tranche d'âge
Nb_client_genre_âge = df_merge_filtré.
 ↪groupby(['sex','tranche_age'],observed=True) [['client_id']].nunique().
 ↪reset_index()

#Visualisation
plt.figure(figsize=(10,6))
sns.barplot(
```

```
    data=Nb_client_genre_âge ,
    x="tranche_age",
    y="client_id",
    hue="sex"
)
plt.ylabel("Nombre de clients")
plt.xlabel("Tranche d'âge")
plt.title("Nombre de clients par Genre et Tranche d'Âge")
plt.legend(title='Genre')
plt.tight_layout()
plt.show()
```



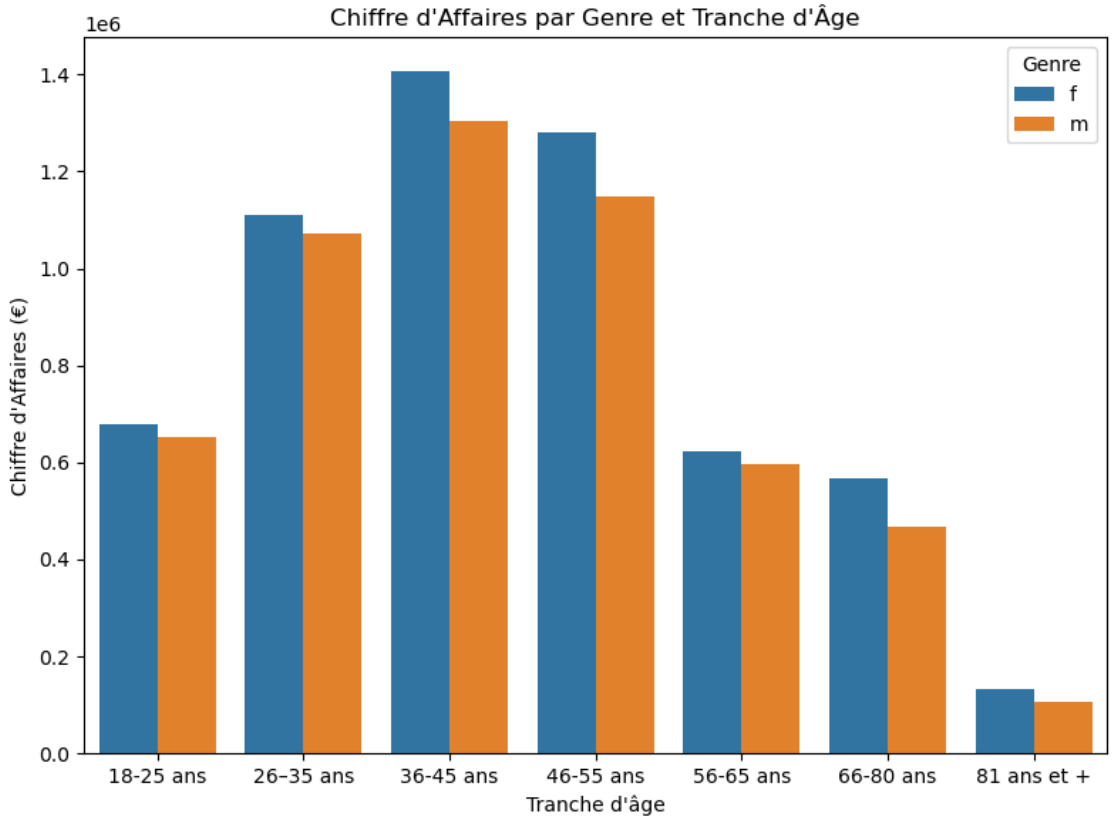
Dans l'ensemble, il y a autant de femmes que d'hommes mais les femmes sont légèrement plus nombreuses parmi la clientèle et ce, quelque que soit la tranche d'âge. La majorité de la clientèle se situe entre 36 et 55 ans.

Les personnes de 80 ans et plus sont les moins représentés.

1.17 3.13 Chiffre d'affaires par genre et tranche d'âge

[88]:
```
#Calcul du CA par genre
CA_genre = df_merge_filtré.groupby(['sex','tranche_age'],observed=True)␣
 ↪['price'].sum().reset_index()

#Visualisation
plt.figure(figsize=(8,6))
```

```
sns.barplot(
    data=CA_genre,
    x="tranche_age",
    y="price",
    hue="sex",
)
plt.ylabel("Chiffre d'Affaires (€)")
plt.xlabel("Tranche d'âge")
plt.title("Chiffre d'Affaires par Genre et Tranche d'Âge")
plt.legend(title='Genre')
plt.tight_layout()
plt.show()
```



Les clients entre 36 et 45 ans gènèrent le plus de chiffres d'affaires. Les personnes de 80 ans et plus apportent le moins de chiffres d'affaires à la librairie.

Etape 4. Analyse des corrélations

1.18 4.2 Lien entre l'âge des clients et le montant total des achats

Nous avons 2 variables de type quantitatives. Reste à savoir si nous devons appliquer un test paramétrique (distribution suivant une loi normale) ou non paramétrique (distribution qui ne suit

pas une loi normale) en vérifiant par un test de normalité.

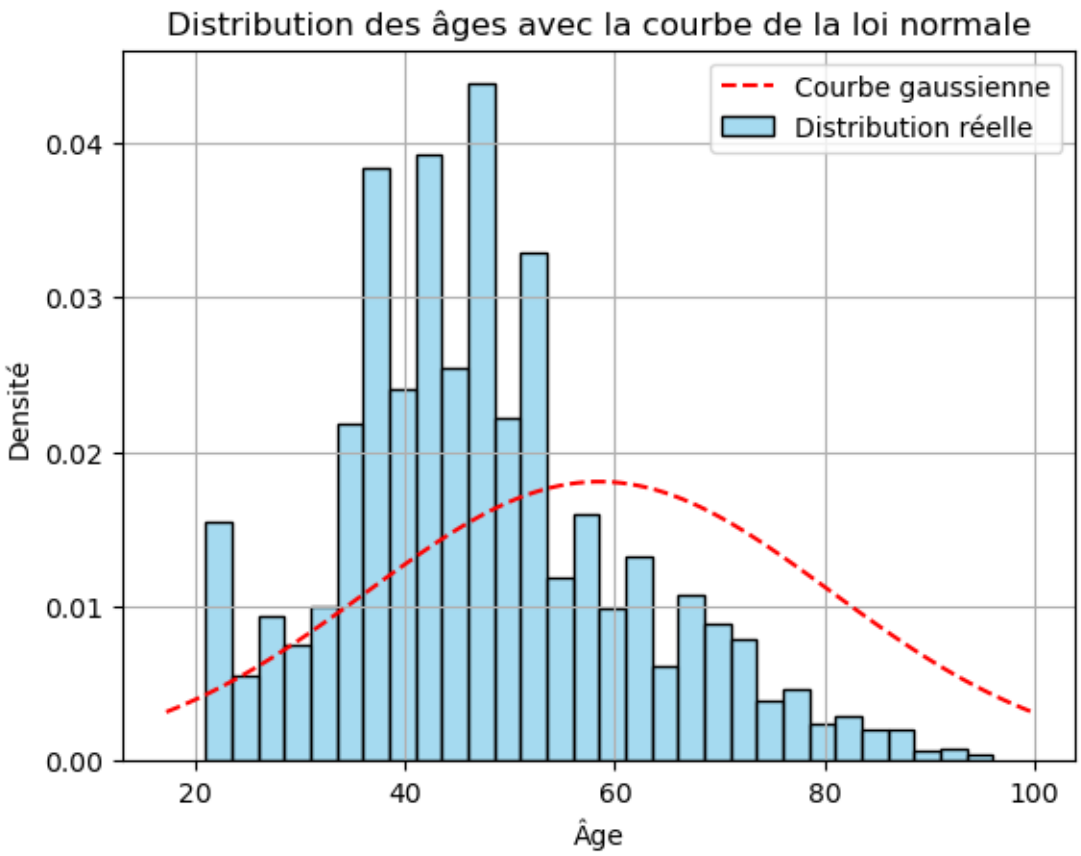
[89]:
```
#Dataframe des montants d'achat par âge
montant_age = df_merge_filtré.groupby('âge').agg({'price':'sum'}).reset_index().
 ↪sort_values(by='âge',ascending=True)
montant_age.head()
```

[89]:

	âge	price
0	21	616418.15
1	22	180680.89
2	23	186791.83
3	24	181233.57
4	25	167388.08

1.18.1 4.2.1 Visualisation de la distribution des âges

[90]:
```
# 1. Calcul des paramètres de la loi normale pour **l'âge**
moyenne_age = montant_age['âge'].mean()
ecart_type_age = montant_age['âge'].std()

# 2. Créer l'histogramme avec la courbe de densité normale
sns.histplot(df_merge_filtré['âge'], bins=30, kde=False, stat="density",␣
 ↪color='skyblue', label="Distribution réelle")

# 3. Superposer la courbe normale théorique
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, moyenne_age, ecart_type_age)
plt.plot(x, p, 'r--', label='Courbe gaussienne')

# 4. Ajouter légende et titre
plt.title('Distribution des âges avec la courbe de la loi normale')
plt.xlabel('Âge')
plt.ylabel('Densité')
plt.legend()
plt.grid(True)
plt.show()
```



Formulation des hypothèses: * Hypothèse nulle (H0) : Les données suivent une distribution normale.

- Hypothèse alternative (H1) : Les données ne suivent pas une distribution normale. Elle est retenue si les données fournissent des preuves suffisantes pour rejeter l'hypothèse nulle.

1.18.2 4.2.2 Test statistique de la normalité de la distribution des âges (Shapiro-wilk)

Test de shapiro-wilk (W):

- si W est proche de 1 = parfaite adéquation avec une distribution normale.
- si W est proche de 0 = mauvaise adéquation avec une distribution normale.

Valeur p:

- si p > 0.05 = pas de raison de rejeter H0 (la normalité de la distribution)
- si p < 0.05 = rejetter l'hypothèse H0

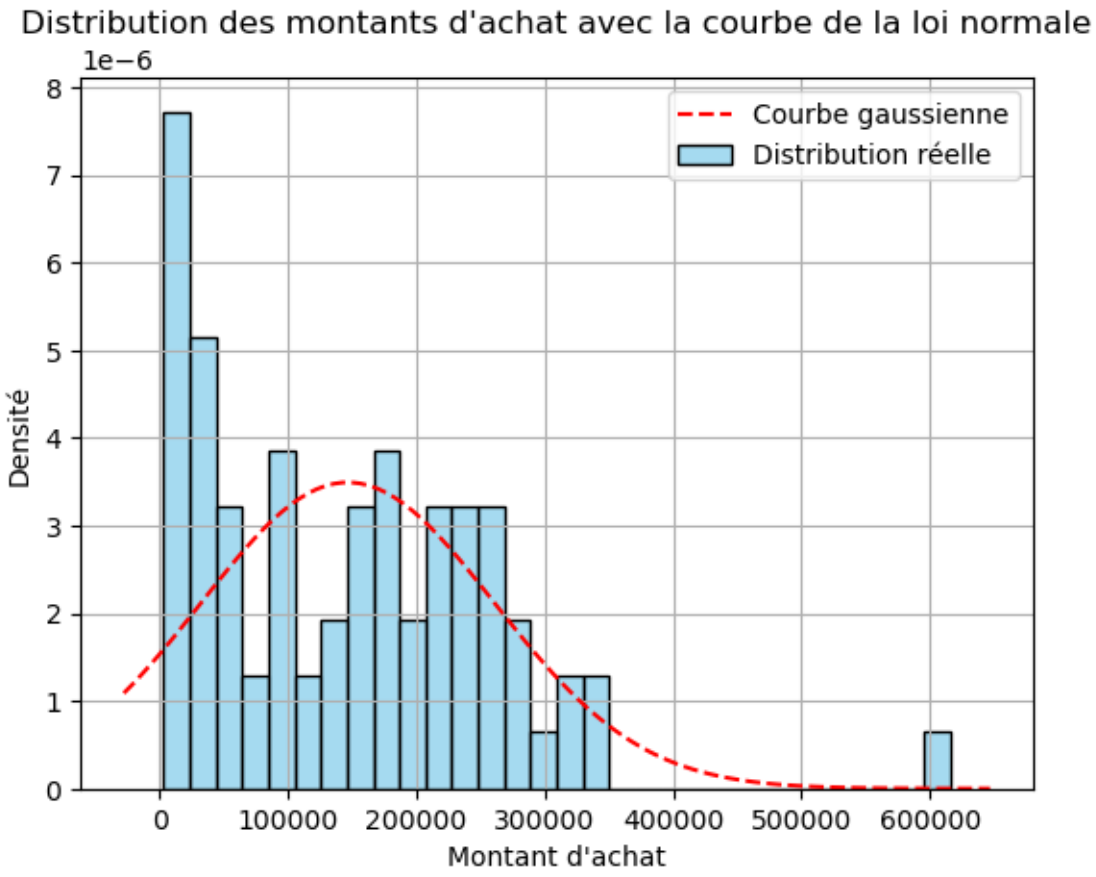
[91]:
```
# Application du test de Shapiro-Wilk pour tester la normalité des âges

 #1. J'extrais la colonne des âges
```

```
age = montant_age['âge'].head(5000)  #Je prends 5000 lignes en exemple car ce␣
 ↪test ne peut pas s'utiliser sur un échantillon de plus de 5000␣
 ↪enregistrements

#2. J'effectue le test de Shapiro-Wilk
stat, p_value = shapiro(age)

 #3. Affichage des résultats
print(f"Statistique du test de Shapiro-Wilk : {stat}")
print(f"Valeur p : {p_value}")

if p_value > 0.05:
    print ("La distribution suit une loi normale")
else: print ("La distribution ne suit pas une loi normale")
```

```
Statistique du test de Shapiro-Wilk : 0.9549230726696206
Valeur p : 0.008752885621051844
La distribution ne suit pas une loi normale
```

1.18.3 4.2.3 Visualisation de la distribution des montants d'achat

[92]:
```
# 1. Calcul des paramètres de la loi normale pour le **montant des achats**
moyenne_prix = montant_age['price'].mean()
ecart_type_prix = montant_age['price'].std()

# 2. Créer l'histogramme avec la courbe de densité normale
sns.histplot(montant_age['price'], bins=30,kde=False, stat="density",␣
 ↪color='skyblue', label="Distribution réelle")

# 3. Superposer la courbe normale théorique
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, moyenne_prix, ecart_type_prix)
plt.plot(x, p, 'r--', label='Courbe gaussienne')

# 4. Ajouter légende et titre
plt.title("Distribution des montants d'achat avec la courbe de la loi normale")
plt.xlabel("Montant d'achat")
plt.ylabel('Densité')
plt.legend()
plt.grid(True)
plt.show()
```



1.18.4 4.2.4 Test statistique de Shapiro-Wilk pour vérifier la normalité des montants d'achat

[93]:
```
# Application du test de Shapiro-Wilk pour tester la normalité des **montants d'achat**

 #1. J'extrais la colonne des prix
price = montant_age['price'].head(5000)  #Je prends 5000 lignes en exemple

 #2. J'effectue le test de Shapiro-Wilk
stat, p_value = shapiro(price)

 #3. Affichage des résultats
print(f"Statistique du test de Shapiro-Wilk : {stat}")
print(f"Valeur p : {p_value}")

 #Interprétation des résultats
if p_value > 0.05:
    print ("La distribution suit une loi normale")
```

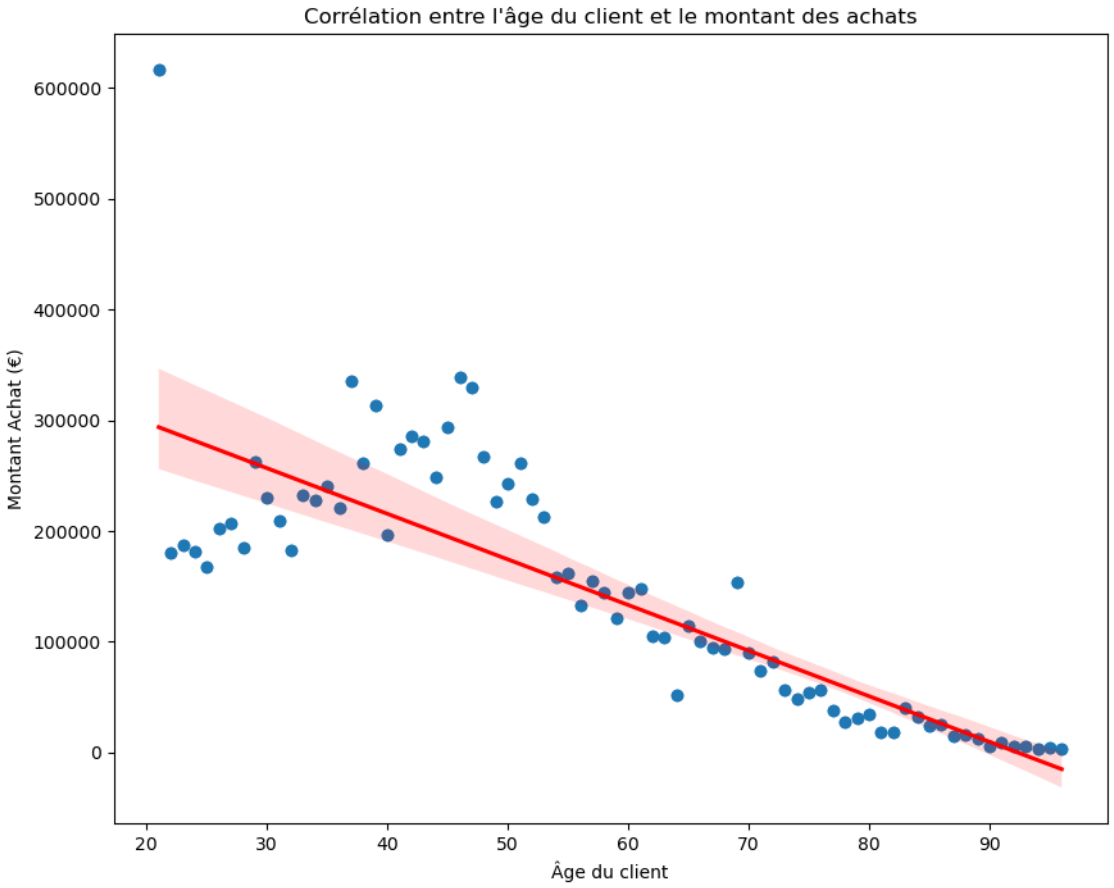
```
else: print ("La distribution ne suit pas une loi normale")
```

```
Statistique du test de Shapiro-Wilk : 0.9124016052922391
Valeur p : 6.473609171793057e-05
La distribution ne suit pas une loi normale
```

On rejette donc l'hypothèse nulle (la distribution suit une loi normale) car la valeur p < 0.05. Donc le test sera non paramétrique

1.18.5 4.2.5 Nuage de point de la corrélation entre l'âge et le montant d'achat

[94]:
```
#Visualisation si la relation est monotone entre l'âge du client et le montant␣
 ↪des achats

plt.figure(figsize=(10,8))

plt.scatter(x=montant_age['âge'], y=montant_age['price'])
sns.regplot(x=montant_age['âge'], y=montant_age['price'], robust=True,␣
 ↪line_kws=dict(color="r"))

plt.title("Corrélation entre l'âge du client et le montant des achats")
plt.xlabel("Âge du client")      # labels après le tracé
plt.ylabel("Montant Achat (€)")

plt.show()
```



La relation entre l'âge et le montant des achats est décroissante. Les clients plus jeunes semble dépenser plus que les clients agés

1.18.6 4.2.6 Test Spearman de la corrélation âge et montant d'achat

Le test de corrélation de Spearman mesure la force et la direction de la relation entre deux variables quantitatives ou ordinales. Il est représenté par le coefficient de corrélation (rho), qui varie entre -1 et 1 :

- $=1$: Corrélation positive parfaite.
- $=-1$: Corrélation négative parfaite.
- $=0$: Aucune corrélation.

La valeur p du test : * valeur p < 0.05 = corrélation réelle. * valeur p > 0.05 = corrélation due au hasard.

[95]:
```
# Calculer le coefficient de corrélation de Spearman et la valeur p
spearman_corr, spearman_p_value =␣
↪spearmanr(montant_age['âge'],montant_age['price'])
```

```
print(f"Coefficient de corrélation de Spearman: {spearman_corr}")
print(f"Valeur p: {spearman_p_value}")
```

```
Coefficient de corrélation de Spearman: -0.8744497607655503
Valeur p: 5.956077505475151e-25
```

En conclusion, il y a une corrélation réelle entre l'âge du client et le montant des achats. Les plus plus jeunes dépensent plus que les plus âgés

1.19 4.3 Lien entre le genre d'un client et les catégories des livres achetés

Le genre du client et la catégorie de livre sont 2 variables qualitatives et indépendantes. Visualisons la fréquence de chaque genre par catégorie pour déterminer quel sera le test à appliquer:

- Si fréquence < 5 = test de Fisher
- Si fréquence > 5 = Test de Chi-2

1.19.1 4.3.1 Table de contingence des catégories de livres et le genre du client

[96]:
```
#Conversion de la colonne 'catég' en type object pour plus de lisibilité
df_merge_filtré.loc[:,'categ'] = df_merge_filtré['categ'].astype(str)
```

```
C:\Users\mende\AppData\Local\Temp\ipykernel_5768\434665494.py:2: FutureWarning:

Setting an item of incompatible dtype is deprecated and will raise in a future
error of pandas. Value '['0' '0' '0' … '2' '2' '2']' has dtype incompatible
with int64, please explicitly cast to a compatible dtype first.
```

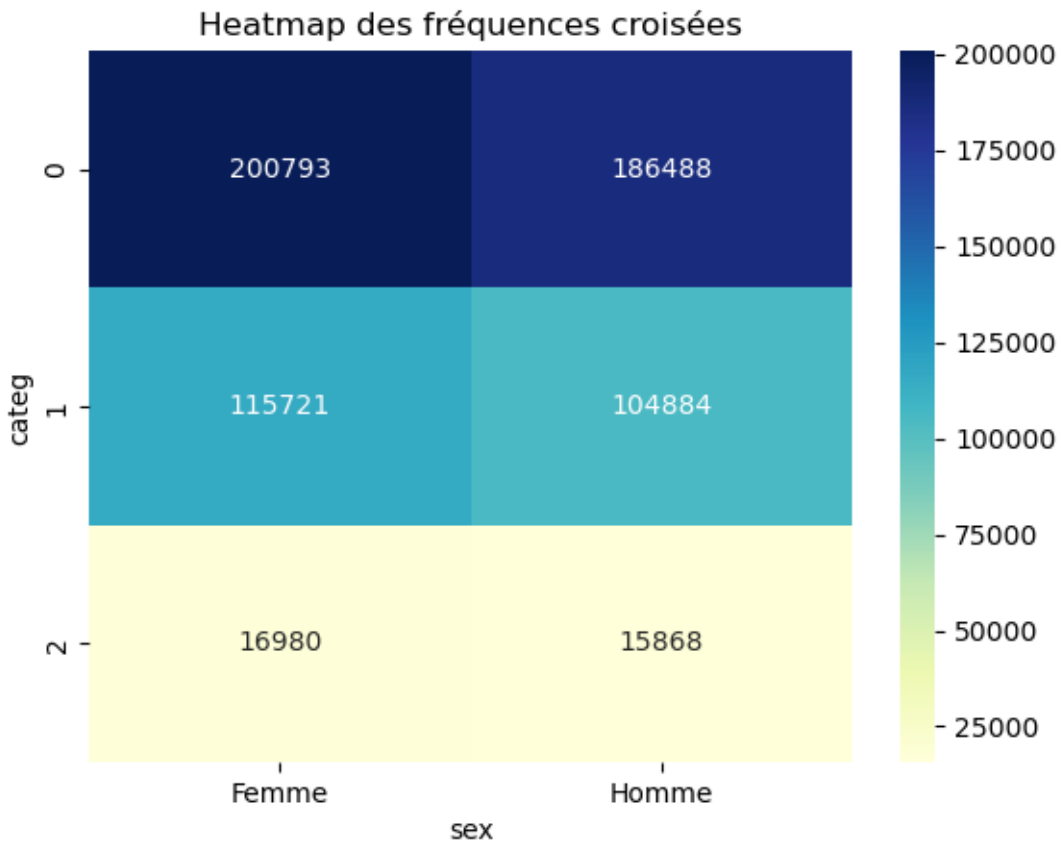
[97]:
```
#Création de la table de contingence pour voir si la fréquence est en dessous␣
↪ou au dessus de 5 pour chaque catégorie
tab_sex_categ = pd.crosstab(df_merge_filtré['categ'], df_merge_filtré['sex'])
tab_sex_categ = tab_sex_categ.rename(columns={'f':'Femme','m':'Homme'})
tab_sex_categ
```

[97]:

sex	Femme	Homme
categ		
0	200793	186488
1	115721	104884
2	16980	15868

1.19.2 4.3.2 Heatmap de la fréquences du genre client pour chaque catégorie

[98]:
```
#Heatmap de la répartition du genre selon la catégorie achetée
sns.heatmap(tab_sex_categ, annot=True, fmt="d", cmap="YlGnBu")
plt.title("Heatmap des fréquences croisées")
plt.show()
```



Nous avons bien plus de 5 fréquences pour chaque catégorie, donc nous allons appliquer le test de Chi-2.

Formulation des hypothèses: * H0 : Le type de livre acheté est indépendant du genre du client.

- H1 : Le type de livre acheté dépend du genre du client.

La valeur p est utilisée pour décider si nous rejetons l'hypothèse nulle H0

Une valeur p :

valeur p $<$ 0.05 = corrélation réelle.

valeur p $>$ 0.05 = corrélation due au hasard.

1.19.3 4.3.3 Test de Chi-2

[99]:
```
#Calcul du test de Chi-2
chi2_stat, p_value, dof, expected = chi2_contingency(tab_sex_categ)

print(f"Statistique Chi-2: {chi2_stat}")
print(f"Valeur p: {p_value}")
print(f"Degrés de liberté: {dof}")
```

```
print("Fréquences attendues:")
print(expected)
```

```
Statistique Chi-2: 22.66856665178056
Valeur p: 1.1955928116587024e-05
Degrés de liberté: 2
Fréquences attendues:
[[201574.89662481 185706.10337519]
 [114822.13191434 105782.86808566]
 [ 17096.97146086  15751.02853914]]
```

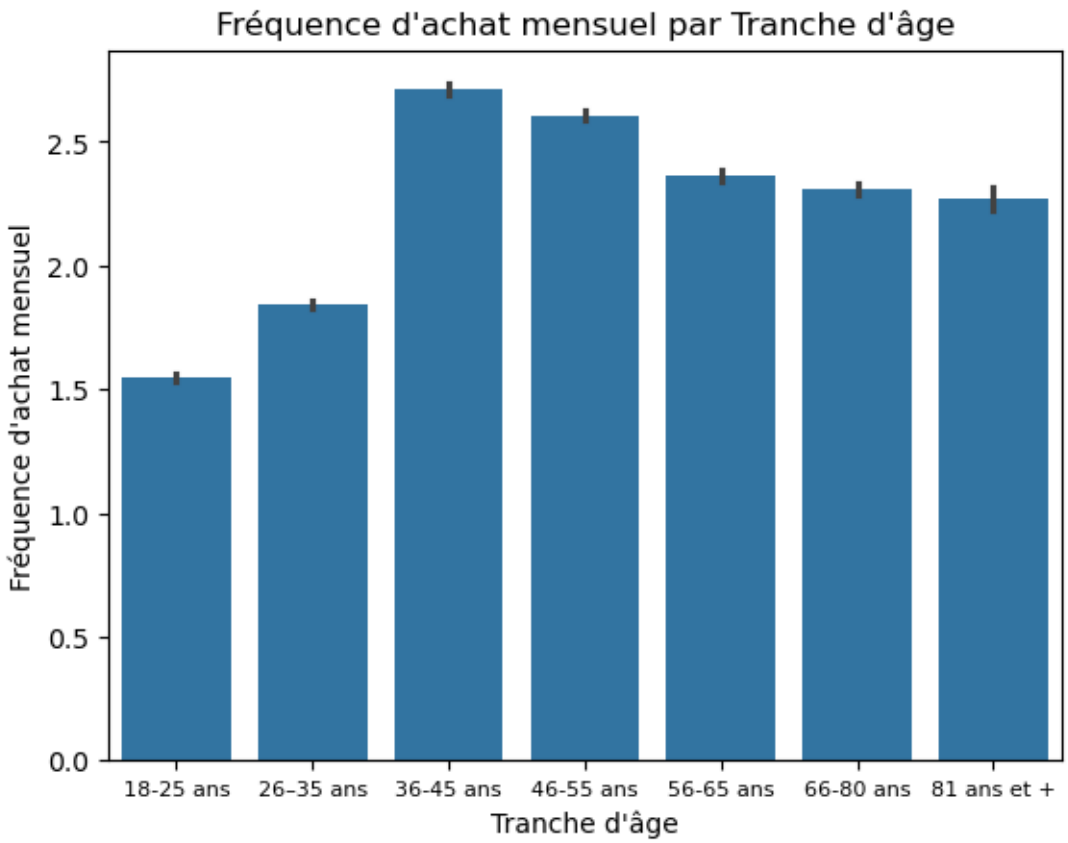
Le test de Chi-2 est utilisé pour déterminer si une distribution observée de données catégorielles diffère d'une distribution attendue. Il compare les fréquences observées dans différentes catégories à des fréquences attendues si les variables étaient indépendantes.Plus elle est grande, plus les différences entre les fréquences observées et attendues sont importantes.

En conclusion, il y a une corrélation réelle entre le genre du client et la catégorie de livre achetés car la valeur p du test est inférieur à 0.05

1.20 4.4 Lien entre l'âge des clients et la fréquence d'achat

Ici, nous avons 2 variables quantitatives et indépendantes. Nous avions vu que la distribution des âges ne suivaient pas de loi normale, vérifions si c'est le cas pour les fréquences d'achat.

[100]:
```
#Création du dataset des fréquences d'achat mensuel selon la tranche d'âge
client_freq_achat_mensuel = df_merge_filtré.groupby(['client_id','mois','âge']).
 ↪agg({'tranche_age':'first','session_id':'nunique'}).reset_index().
 ↪rename(columns={'session_id':"Fréquence d'achat"}).
 ↪sort_values(by='âge',ascending=True)

#Visualisation
sns.barplot(
    client_freq_achat_mensuel,
    x= client_freq_achat_mensuel['tranche_age'],
    y= client_freq_achat_mensuel["Fréquence d'achat"]
)
plt.xlabel("Tranche d'âge")
plt.xticks(rotation=0, fontsize= 8)
plt.ylabel("Fréquence d'achat mensuel")
plt.title("Fréquence d'achat mensuel par Tranche d'âge")
plt.show()
```

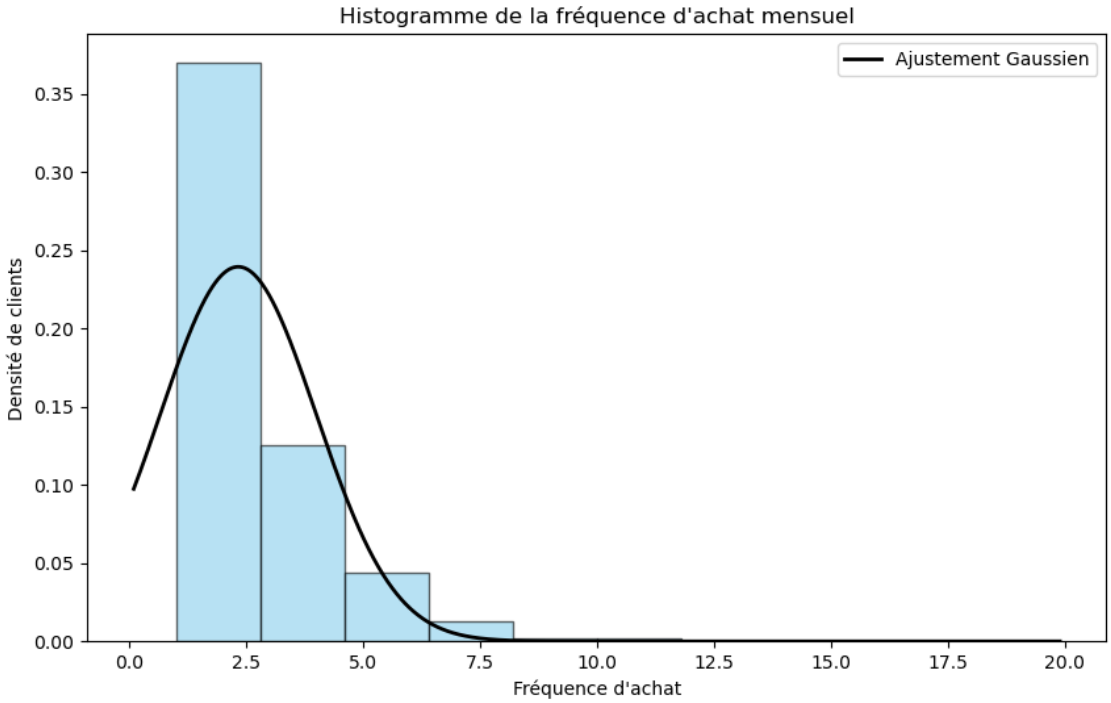


Chaque mois, un client plus âgé a tendance à commander plus régulièrement sur le site que les plus jeunes entre 18 et 35 ans.

1.20.1 4.4.1 Visualisation de la distribution des fréquences d'achat

[101]:
```
#Distribution de la fréquence d'achat

# Histogramme
plt.figure(figsize=(10, 6))
plt.hist(client_freq_achat_mensuel["Fréquence d'achat"], bins=10, density=True,␣
 ↪color='skyblue', edgecolor='black', alpha=0.6)

# Ajustement gaussien
mu, std = norm.fit(client_freq_achat_mensuel["Fréquence d'achat"])   #Calcul la␣
 ↪moyenne et l'écart-type
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, client_freq_achat_mensuel.shape[0])
p = norm.pdf(x, mu, std)

# Tracer la courbe gaussienne
```

```
plt.plot(x, p, 'k', linewidth=2, label='Ajustement Gaussien')

# Ajouter les titres et légendes
plt.title("Histogramme de la fréquence d'achat mensuel")
plt.xlabel("Fréquence d'achat")
plt.ylabel('Densité de clients')
plt.legend()

# Afficher le graphique
plt.show()
```



1.20.2 4.4.2 Test de la normalité shapiro-Wilk de la fréquence d'achat

[102]:
```
# Application du test de Shapiro-Wilk pour tester la normalité de la fréquence␣
↪d'achat

#1. J'extrais la colonne des âges
freq_achat = client_freq_achat_mensuel["Fréquence d'achat"].head(5000)  #Je␣
↪prends 5000 lignes en exemple

#2. J'effectue le test de Shapiro-Wilk
stat, p_value = shapiro(freq_achat)

#3. Affichage des résultats
```

```
print(f"Statistique du test de Shapiro-Wilk : {stat}")
print(f"Valeur p : {p_value}")

#Interprétation des résultats :
if p_value > 0.05 :
    print(f"La données suivent pas une loi normale")
else : print (f"Les données ne suivent pas une loi normale")
```

```
Statistique du test de Shapiro-Wilk : 0.6859536837525061
Valeur p : 2.0441181046246753e-70
Les données ne suivent pas une loi normale
```

1.20.3 4.4.3 Scatter plot entre l'âge et la fréquence d'achat

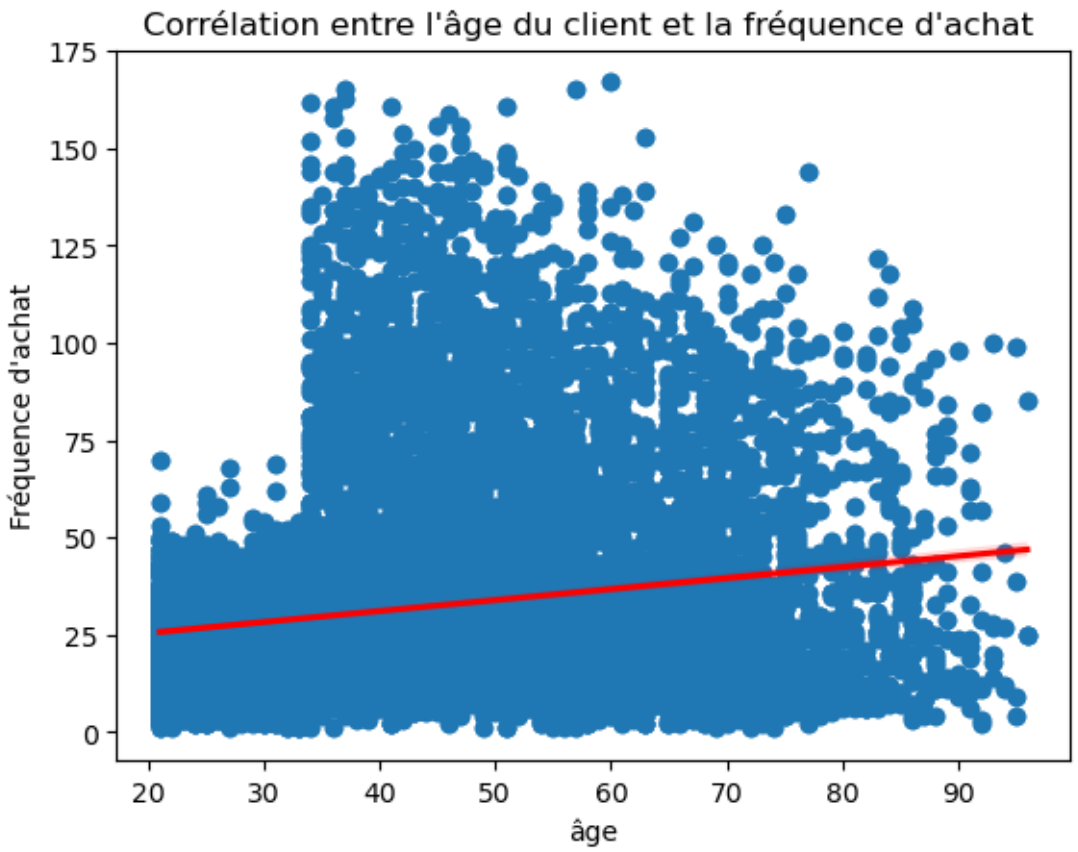
[103]:
```
age_freq_achat = df_merge_filtré.groupby(['client_id','âge']) ['session_id'].
 ↪nunique().reset_index().rename(columns={'session_id':"Fréquence d'achat"})
age_freq_achat.head()
```

[103]:

	client_id	âge	Fréquence d'achat
0	c_1	70	34
1	c_10	69	34
2	c_100	33	5
3	c_1000	59	94
4	c_1001	43	47

[104]:
```
#Visualisation
plt.scatter(
    x= age_freq_achat['âge'],
    y= age_freq_achat["Fréquence d'achat"])

plt.title("Corrélation entre l'âge du client et la fréquence d'achat ")
plt.xlabel("Âge")
plt.ylabel("Fréquence d'Achat")
sns.regplot(x=age_freq_achat['âge'], y=age_freq_achat["Fréquence d'achat"],␣
 ↪robust=True, line_kws=dict(color="r"))
plt.figure(figsize=(10,8))
```

[104]:
```
<Figure size 1000x800 with 0 Axes>
```




```
<Figure size 1000x800 with 0 Axes>
```

1.20.4 4.4.4 Test de Spearman

Etant donnée que la fréquence d'achat ne suit pas une distribution normale, nous allons appliquer le test de Spearman

[105]:
```
# Calculer le coefficient de corrélation de Spearman et la valeur p
spearman_corr, spearman_p_value =␣
 ↪spearmanr(age_freq_achat['âge'],age_freq_achat["Fréquence d'achat"])

print(f"Coefficient de corrélation de Spearman: {spearman_corr}")
print(f"Valeur p: {spearman_p_value}")
```

```
Coefficient de corrélation de Spearman: 0.21196373259671872
Valeur p: 6.629168433162815e-88
```

Il y a une corrélation réelle entre l'âge et la fréquence d'achat. Les plus agés ont tendance à acheter plus régulièrement que les plus jeunes clients.

1.21 4.5 Lien entre l'âge des clients et la taille du panier moyen

Nous avons 2 variables de type quantitatives. Nous allons vérifier si le panier moyen suit bien une distribution normale.

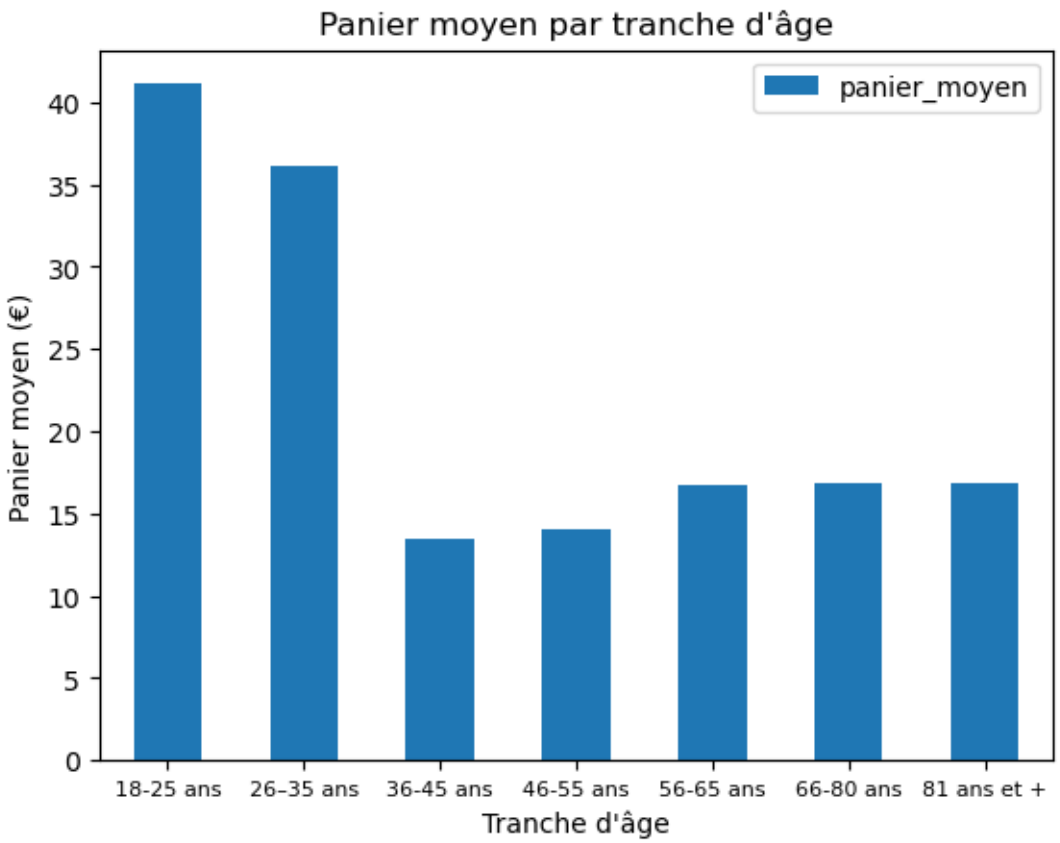
[106]:
```
#Création du dataframe du panier moyen par client
age_client_panier_moyen = df_merge_filtré.groupby('client_id').agg({'âge':
 ↪'first','tranche_age':'first','price':'sum','session_id':'count'})

#Ajout une colonne pour le panier moyen
age_client_panier_moyen['panier_moyen'] =␣
 ↪round(age_client_panier_moyen['price'] /␣
 ↪age_client_panier_moyen['session_id'],2)
#Affichage
age_client_panier_moyen.reset_index().head()
```

[106]:

	client_id	âge	tranche_age	price	session_id	panier_moyen
0	c_1	70	66-80 ans	629.02	43	14.63
1	c_10	69	66-80 ans	1353.60	58	23.34
2	c_100	33	26-35 ans	254.85	8	31.86
3	c_1000	59	56-65 ans	2291.88	126	18.19
4	c_1001	43	36-45 ans	1823.85	103	17.71

[107]:
```
tranche_age_panier_moyen = age_client_panier_moyen.
 ↪groupby('tranche_age',observed=True).agg({'panier_moyen':'mean'}).
 ↪plot(kind='bar')
plt.title("Panier moyen par tranche d'âge")
plt.xlabel("Tranche d'âge")
plt.xticks(rotation=0,fontsize= 8)
plt.ylabel("Panier moyen (€)")
```

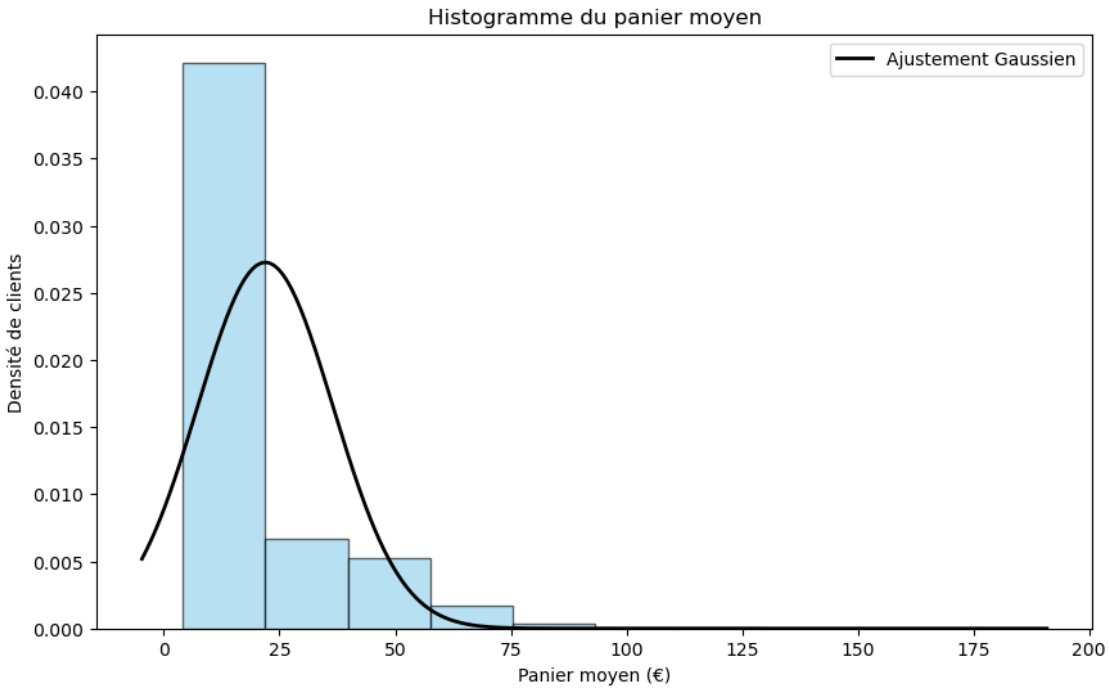
[107]:
```
Text(0, 0.5, 'Panier moyen (€)')
```



1.21.1 4.5.1 Visualisation de la distribution du panier moyen

[108]:
```
#Distribution du panier moyen

# Histogramme
plt.figure(figsize=(10, 6))
plt.hist(age_client_panier_moyen["panier_moyen"], bins=10, density=True,␣
 ↪color='skyblue', edgecolor='black', alpha=0.6)

# Ajustement gaussien
mu, std = norm.fit(age_client_panier_moyen["panier_moyen"])   #Calcul la␣
 ↪moyenne et l'écart-type
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, age_client_panier_moyen.shape[0])
p = norm.pdf(x, mu, std)

# Tracer la courbe gaussienne
plt.plot(x, p, 'k', linewidth=2, label='Ajustement Gaussien')

# Ajouter les titres et légendes
```

```
plt.title("Histogramme du panier moyen")
plt.xlabel("Panier moyen (€)")
plt.ylabel('Densité de clients')
plt.legend()

# Afficher le graphique
plt.show()
```



1.21.2 4.5.2 Test de la normalité de la distribution du panier moyen

Formulation des hypothèses: * Hypothèse nulle (H0) : Les données suivent une distribution normale. * Hypothèse alternative (H1) : Les données ne suivent pas une distribution normale. Elle est retenue si les données fournissent des preuves suffisantes pour rejeter l'hypothèse nulle.

[109]:
```
# Application du test de Shapiro-Wilk pour tester la normalité du panier moyen

 #1. J'extrais la colonne du panier moyen
panier_moyen = age_client_panier_moyen["panier_moyen"].head(5000)  #Je prends␣
 ↪5000 lignes en exemple

 #2. J'effectue le test de Shapiro-Wilk
stat, p_value = shapiro(panier_moyen)

 #3. Affichage des résultats
print(f"Statistique du test de Shapiro-Wilk : {stat}")
```

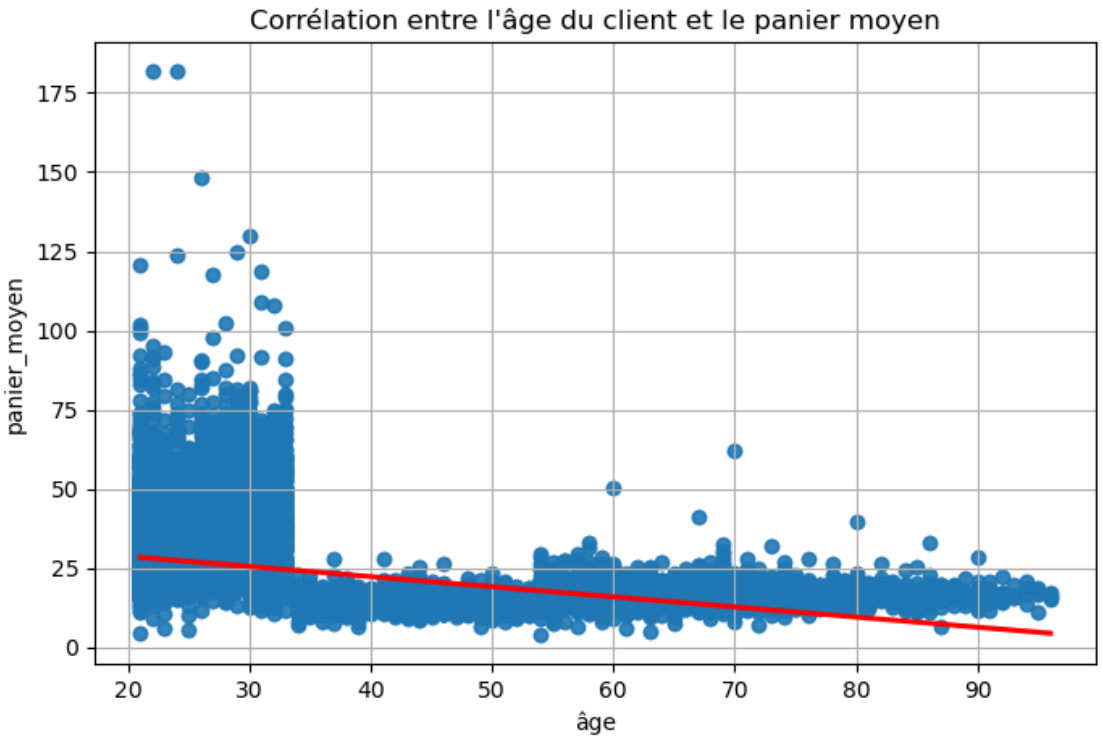
```
print(f"Valeur p : {p_value}")

#Interprétation des résultats :
if p_value > 0.05 :
    print(f"La données suivent pas une loi normale")
else : print (f"Les données ne suivent pas une loi normale")
```

```
Statistique du test de Shapiro-Wilk : 0.6831194158291327
Valeur p : 1.3445636759125993e-70
Les données ne suivent pas une loi normale
```

Nous rejettons l'hypothèse nulle donc nous allons devoir utiliser un test non paramétrique

1.21.3 4.5.3 Visualisation de la corrélation entre l'âge et le panier moyen

[110]:
```
#Visualisation
plt.figure(figsize=(8, 5))
plt.
  ↪scatter(age_client_panier_moyen['âge'],age_client_panier_moyen['panier_moyen'],␣
  ↪alpha=0.5)
plt.title("Corrélation entre l'âge du client et le panier moyen")
plt.xlabel("Âge du client")
plt.ylabel("Panier moyen (€)")
plt.grid(True)
sns.regplot(x=age_client_panier_moyen['âge'],␣
  ↪y=age_client_panier_moyen['panier_moyen'],robust=True,␣
  ↪line_kws=dict(color="r"))
plt.show()
```



Il semble que plus l'âge augmente et moins le panier moyen est élevé.

1.21.4 4.5.4 Test de Spearman

[111]:
```
# Calculer le coefficient de corrélation de Spearman et la valeur p
spearman_corr, spearman_p_value =␣
↪spearmanr(age_client_panier_moyen['âge'],age_client_panier_moyen["panier_moyen"])

print(f"Coefficient de corrélation de Spearman: {spearman_corr}")
print(f"Valeur p: {spearman_p_value}")
```

```
Coefficient de corrélation de Spearman: -0.32587401420827466
Valeur p: 8.202257290884015e-212
```

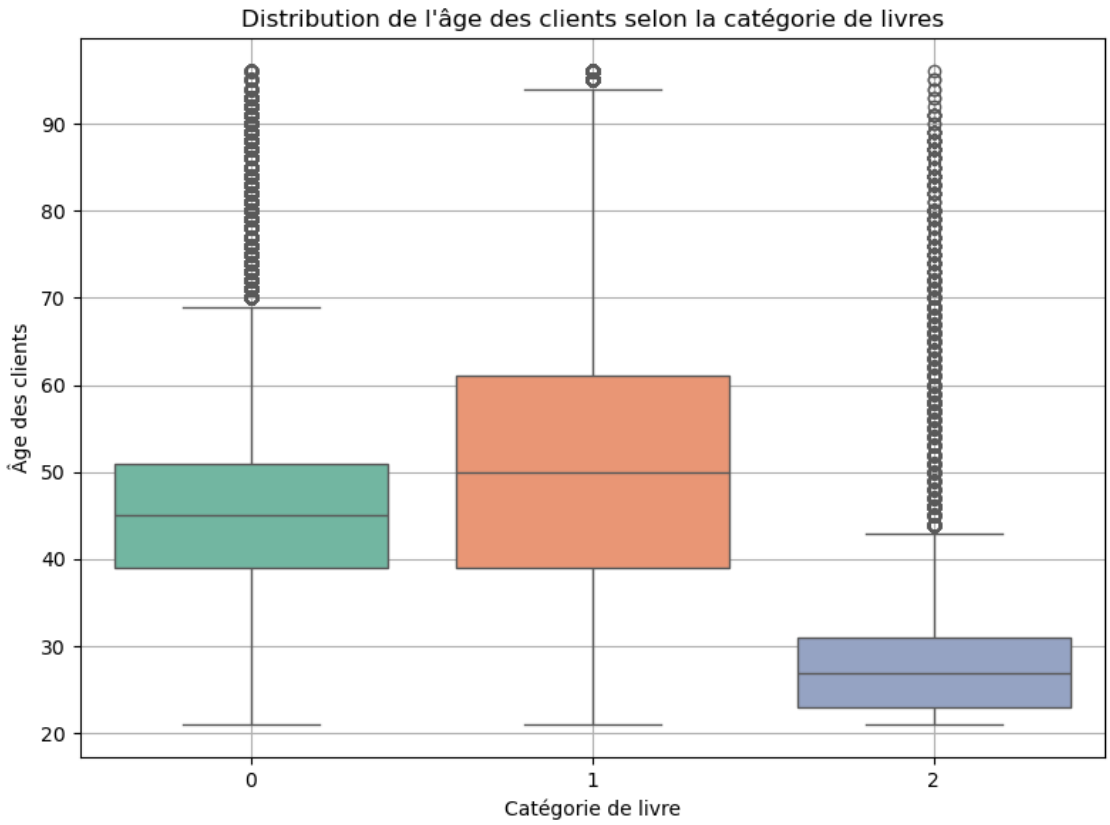
La corrélation est statistiquement significative entre l'âge du client et le panier moyen. Les jeunes dépensent plus à chaque commande que les clients plus âgés.

1.22 4.6 Lien entre l'âge des clients et la catégorie des livres achetés

Nous avons une variable quantitative et qualitative donc une approche mixte. Dans ce cas, nous devons vérifier si pour le nombre de modalités pour la catégorie de livre est 2 ou plus . Ici, nous avons 3 catégories de livre donc nous allons soit appliquer un test ANOVA ou test Kruskal Wallis.

1.22.1 4.6.1 Visualisation de la répartition de l'âge pour chaque catégorie des livres

[112]:
```
#Visualisation
plt.figure(figsize=(8, 6))
sns.boxplot(x='categ', y='âge', data=df_merge_filtré, palette='Set2',␣
 ↪hue='categ')

plt.title("Distribution de l'âge des clients selon la catégorie de livres")
plt.xlabel("Catégorie de livre")
plt.ylabel("Âge des clients")
plt.grid(True)
plt.tight_layout()
plt.show()
```

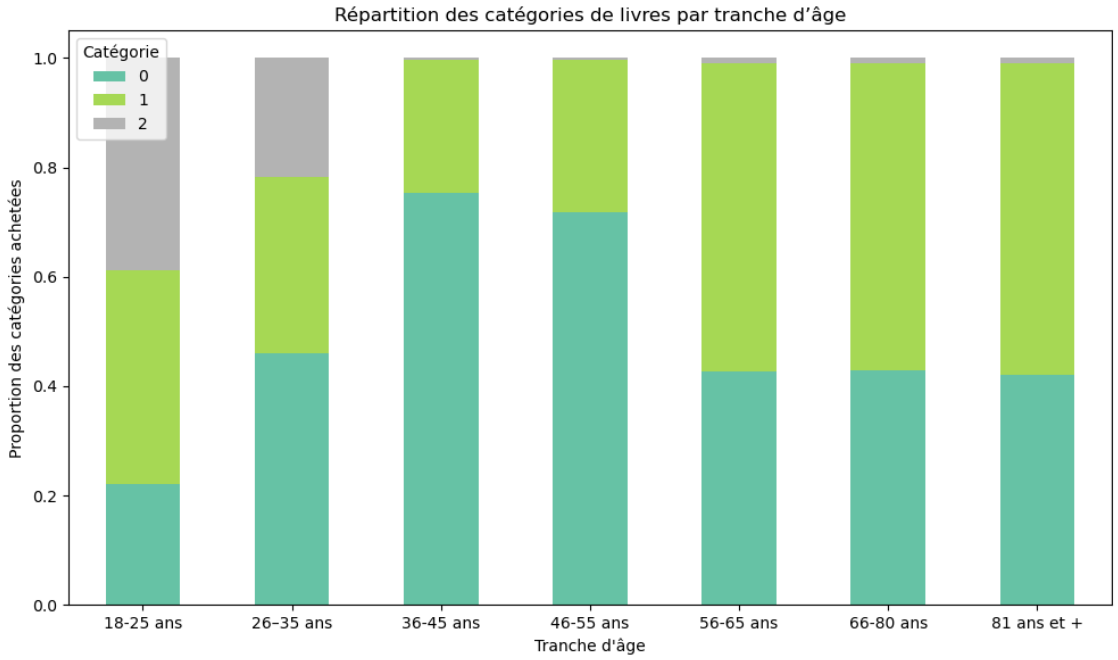


Chaque catégorie de livre a son public selon la tranche d'âge selon ce boxplot

[113]:
```
#Création d'un tableau de contingence du nombre de livres achetés dans chaque␣
 ↪catégorie par tranche d'âge
tab_tranche_age_categ = pd.crosstab(df_merge_filtré['tranche_age'],␣
 ↪df_merge_filtré['categ'])
tab_tranche_age_categ
```

[113]:
```
categ              0      1      2
tranche_age
18-25 ans       7431  13135  13040
26-35 ans      35870  25121  16986
36-45 ans     153421  49473    764
46-55 ans     127292  49523    713
56-65 ans      30944  40773    646
66-80 ans      26419  34581    566
81 ans et +     5904   7999    133
```

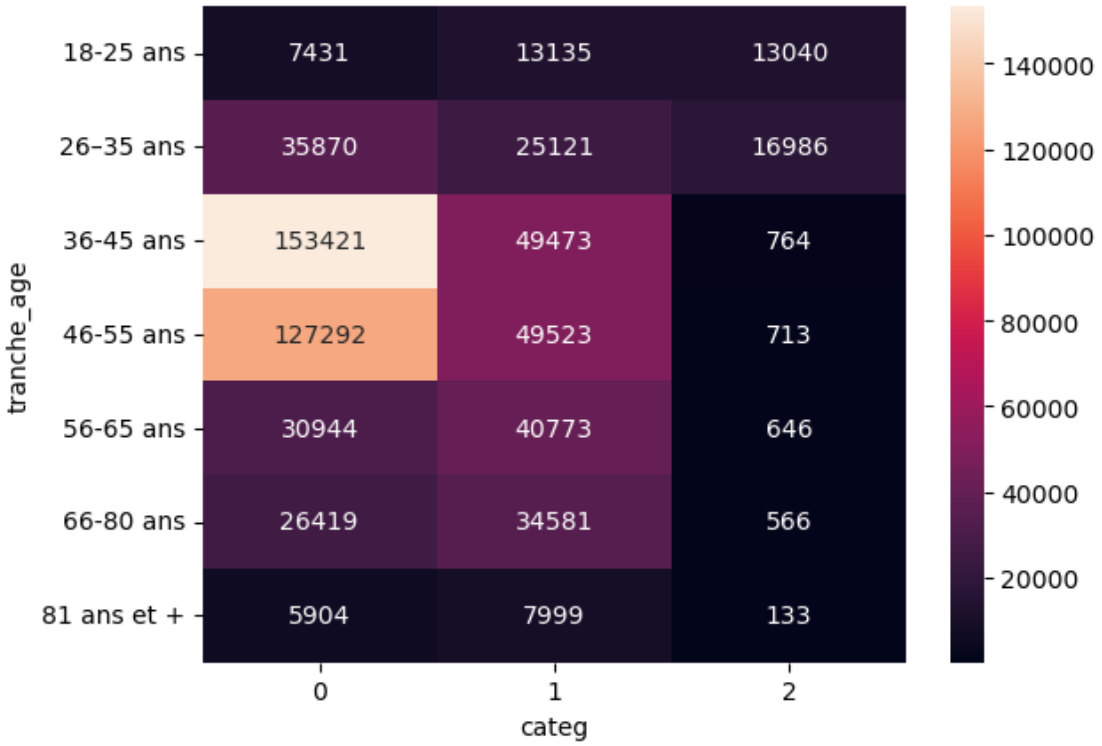
[114]:
```
table_pct = tab_tranche_age_categ.div(tab_tranche_age_categ.sum(axis=1),axis=0)
 ↪ # Pourcentage par ligne
table_pct.plot(kind='bar', stacked=True, colormap='Set2', figsize=(10,6), rot=0)

plt.title("Répartition des catégories de livres par tranche d'âge")
plt.xlabel("Tranche d'âge")
plt.ylabel("Proportion des catégories achetées")
plt.legend(title="Catégorie")
plt.tight_layout()
plt.show()
```



On remarque que selon les tranches d'âge, des catégories de livre sont plus achetés que d'autres. Seulement la tranche d'âge de 18-25 achète différente catégorie de livre.

[115]:
```
#Heatmap
sns.heatmap(tab_tranche_age_categ, annot=True,fmt=".0f")
```

[115]: <Axes: xlabel='categ', ylabel='tranche_age'>



Avec ce heatmap, on remarque qu'il y a une faible de corrélation entre la tranche d'âge et la catégorie de livre acheté. Exception pour les tranches d'âge 36-45ans et 46-55ans qui achète majoritairement des livres de catégorie 0.

1.22.2 4.6.2 Test de Kruskal Wallis

Etant donné que les âges ne suivent pas une loi normale, alors on va appliquer un test non paramétrique de Kruskal Wallis.

[116]:
```
#Création d'une liste d'âge pour chaque catégorie de livre acheté pour le test␣
↪de Kruskal
cat_0 = df_merge_filtré[df_merge_filtré['categ'] == '0']['âge']
cat_1 = df_merge_filtré[df_merge_filtré['categ'] == '1']['âge']
cat_2 = df_merge_filtré[df_merge_filtré['categ'] == '2']['âge']
```

Formulation d'hypothèse: * H0 (hypothèse nulle) : les distributions d'âge sont identiques entre les catégories.

- H1 (hypothèse alternative) : au moins une catégorie a une distribution d'âges différente.

[117]:
```
# Test de Kruskal-Wallis
h_stat, p_value = kruskal(cat_0, cat_1, cat_2)
```

```
print(f"Statistique du test de Kruskal:{h_stat}")
print(f"Valeur p:{p_value}")
```

```
Statistique du test de Kruskal:71359.73412120914
Valeur p:0.0
```

Le test de Kruskal-Wallis révèle qu'il n'y a pas de corrélation entre l'âge et les catégories de livres achetés. Les préférences de catégories ne sont pas influencées par l'âge du client.

Recommandations

Corrélation réelle entre l'âge du client et le montant des achats / panier moyen : Les plus plus jeunes ont un panier moyen plus élevé que les plus âgés

–> Pour les jeunes clients, proposer des packs de 2 livres premium. Et récompenser leur achat en donnant des avantages sur les nouveautés.

Corrélation significative entre le genre du client et la catégorie de livre achetée

–> Selon le genre du client qui se connecte sur son compte, lui proposer dès la page d'accueil les catégories susceptibles de l'intéresser. Par le biais de newsletters,

Il y a une corrélation réelle entre l'âge et la fréquence d'achat. Les plus agés ont tendance à acheter plus souvent que les plus jeunes clients.

–> Proposer des avantages/réductions de fidélité ou proposer des abonnements mensuels/annuels.

Il n'y a pas de dépendance statistique entre l'âge et la catégorie de livre achetée.

–> Peu importe l'âge, proposer toutes les catégories de livres. Cibler les catégories en fonction du genre plutôt que de l'âge.

[118]:
```
#Création du fichier excel
df_merge.to_excel("ventes_librairie.xlsx", index=False)
```