Analysez les ventes d'une librairie avec R ou Python - Projet 6-Copy4

October 2, 2024

```
<h1>Analysez les ventes d'une librairie avec R ou Python : Lapage </h1>
```

```
<h2 style="font-weight: bold; color: #001F3F;">Etape 1 - Importation des librairies et chargeme
```

1.1 - Importation des librairies

[1]:
```
#Importation des librairies
import pandas as pd
import random
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import locale
import numpy as np
import scipy.stats as stats
from datetime import datetime
import plotly.express as px
from plotly.io import write_image
import seaborn as sns
from scipy.stats import chi2_contingency
from scipy.stats import f_oneway
```

1.2 - Création des fonctions

[2]:
```
def verifier_unicite(df, colonnes_cles):
    # Vérifie si des doublons sont présents dans la combinaison des colonnes␣
 ↪spécifiées
    doublons = df.duplicated(subset=colonnes_cles, keep=False)
    # Compte le nombre de doublons
    nombre_doublons = doublons.sum()
    # Affiche le résultat
    if nombre_doublons == 0:
        print(f"La combinaison des clés {colonnes_cles} est unique dans le␣
 ↪DataFrame.")
    else:
        print(f"La combinaison des clés {colonnes_cles} n'est pas unique dans␣
 ↪le DataFrame et il y a {nombre_doublons} doublons.")
```

1.3 - chargement des fichiers

[3]:
```
#Importation des fichiers
customers = pd.read_csv('/Users/Bouboule/Documents/Projet 6/DAN-P6-donnees/
 ↪customers.csv')
products = pd.read_csv('/Users/Bouboule/Documents/Projet 6/DAN-P6-donnees/
 ↪products.csv')
transactions = pd.read_csv('/Users/Bouboule/Documents/Projet 6/DAN-P6-donnees/
 ↪transactions.csv')
```

<h2 style="font-weight: bold; color: #001F3F;">Etape 2 - Analyse du fichier customers</h2>

2.1 - Analyse exploratoire

[4]:
```
#Consulter les infos du dataframe
customers.info()
```

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

[5]:
```
#Affichage des 5 premières lignes de la table
customers.head()
```

[5]:

	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

2.2 - Vérification de l'unicité de la clé

[6]:
```
verifier_unicite(customers, 'client_id')
```

```
La combinaison des clés client_id est unique dans le DataFrame.
```

2.3 - Vérification des incohérences

2.3.1 - Vérification de la colonne sex

[7]:
```
# Vérifier s'il y a des valeurs différentes de 'f' ou 'm' dans la colonne 'sex'
valeurs_non_valides = customers.loc[~customers['sex'].isin(['f', 'm']), 'sex']
if valeurs_non_valides.empty:
    print("Toutes les valeurs de la colonne 'sex' sont valides (soit 'f' soit␣
 ↪'m').")
```

```
else:
    print("Il y a des valeurs différentes de 'f' ou 'm' dans la colonne 'sex':")
    print(valeurs_non_valides)
```

```
Toutes les valeurs de la colonne 'sex' sont valides (soit 'f' soit 'm').
```

2.3.2 - Vérification de la colonne birth

[8]:
```
# Vérifier s'il y a des valeurs négatives, inférieures à 1920 ou supérieures à␣
 ↪2023 dans la colonne 'birth'
valeurs_non_valides = customers.loc[(customers['birth'] < 0) |␣
 ↪(customers['birth'] < 1920) | (customers['birth'] > 2023), 'birth']
if valeurs_non_valides.empty:
    print("Toutes les valeurs de la colonne 'birth' sont valides.")
else:
    print("Il y a des valeurs négatives, inférieures à 1920, ou supérieures à␣
 ↪2023 dans la colonne 'birth':")
    print(valeurs_non_valides)
```

```
Toutes les valeurs de la colonne 'birth' sont valides.
```

2.3.3 - Vérification du format de l'id

[9]:
```
#Définir le pattern de l'id
pattern = r'^c_\d+$'
# Check si les ids match avec le pattern
is_valid_id = customers['client_id'].str.match(pattern, na=False).all()
if is_valid_id:
    print("Tous les identifiants suivent le nouveau format attendu.")
else:
    # Afficher les ids invalides
    invalid_ids = customers.loc[~customers['client_id'].str.match(pattern,␣
 ↪na=False), 'client_id']
    print("Certains identifiants ne suivent pas le format attendu:")
```

```
Certains identifiants ne suivent pas le format attendu:
```

```
<h2 style="font-weight: bold; color: #001F3F;">Etape 3 - Analyse du fichier products</h2>
```

3.1 - Analyse exploratoire

[10]:
```
#Consulter les infos du dataframe
products.info()
```

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

```
 2   categ    3287 non-null   int64
dtypes: float64(1), int64(1), object(1)
memory usage: 77.2+ KB
```

[11]:
```
products.head()
```

[11]:
```
  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
```

3.2 - Vérification de l'unicité de la clé

[12]:
```
verifier_unicite(products, 'id_prod')
```

```
La combinaison des clés id_prod est unique dans le DataFrame.
```

3.3 - Vérification des incohérences

[13]:
```
# Filtrer pour trouver les prix négatifs
prix_negatifs = products[products['price'] < 0]
# Vérifier s'il y a des prix négatifs
if not prix_negatifs.empty:
    print("Il y a des prix avec une valeur négative dans la colonne price.")
    print(prix_negatifs)
else:
    print("Il n'y a pas de prix avec une valeur négative dans la colonne price.
 ↪")
```

```
Il y a des prix avec une valeur négative dans la colonne price.
    id_prod  price  categ
731     T_0   -1.0      0
```

3.3 - Suppression des valeurs incohérentes

[14]:
```
# Sélectionner les indices des lignes où le prix est négatif
prix_negatifs = products[products['price'] < 0].index
# Supprimer les lignes avec des prix négatifs
products = products.drop(prix_negatifs)
```

[15]:
```
#Consulter les infos du dataframe
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3286 entries, 0 to 3286
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: 102.7+ KB
```

<h2 style="font-weight: bold; color: #001F3F;">Etape 4 - Analyse du fichier transactions</h2>

4.1 - Analyse exploratoire

[16]:
```
#Consulter les infos du dataframe
transactions.info()
```

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

[17]:
```
transactions.head()
```

[17]:

	id_prod	date	session_id	client_id
0	0_1518	2022-05-20 13:21:29.043970	s_211425	c_103
1	1_251	2022-02-02 07:55:19.149409	s_158752	c_8534
2	0_1277	2022-06-18 15:44:33.155329	s_225667	c_6714
3	2_209	2021-06-24 04:19:29.835891	s_52962	c_6941
4	0_1509	2023-01-11 08:22:08.194479	s_325227	c_4232

4.2 - Vérification des incohérences

4.3.1 - Vérification du format des ids

[18]:
```
#Définir le pattern de l'id
pattern = r'^c_\d+$'
# Check si les ids match avec le pattern
is_valid_id = transactions['client_id'].str.match(pattern, na=False).all()
if is_valid_id:
    print("Tous les identifiants suivent le nouveau format attendu.")
else:
    # Afficher les ids invalides
    invalid_ids = transactions.loc[~transactions['client_id'].str.
 ↪match(pattern, na=False), 'client_id']
    print("Certains identifiants ne suivent pas le format attendu:")
    print(invalid_ids)
```

```
Certains identifiants ne suivent pas le format attendu:
3019      ct_0
5138      ct_0
9668      ct_1
10728     ct_0
15292     ct_0
          …
657830    ct_0
662081    ct_1
670680    ct_1
671647    ct_1
679180    ct_1
Name: client_id, Length: 200, dtype: object
```

[19]:
```python
#Définir le pattern de l'id
pattern = r'^s_\d+$'
# Check si les ids match avec le pattern
is_valid_id = transactions['session_id'].str.match(pattern, na=False).all()
if is_valid_id:
    print("Tous les identifiants suivent le format attendu.")
else:
    # Afficher les ids invalides
    invalid_ids = transactions.loc[~transactions['session_id'].str.
 ↪match(pattern, na=False), 'session_id']
    print("Certains identifiants ne suivent pas le format attendu:")
    print(invalid_ids)
```

```
Tous les identifiants suivent le format attendu.
```

[20]:
```python
# Définir le pattern
pattern = r'^\d+_\d+$'
# Vérifier si les id prod correspondent au pattern
is_valid_id = transactions['id_prod'].str.match(pattern, na=False)
if is_valid_id.all():
    print("Tous les identifiants suivent le nouveau format attendu.")
else:
    invalid_ids = transactions.loc[~is_valid_id, 'id_prod']
    print("Certains identifiants ne suivent pas le format attendu:")
    print(invalid_ids)

    num_invalid_ids = is_valid_id.value_counts()[False]

print(f"Nombre de lignes avec un format non valide : {num_invalid_ids}")
print(transactions.loc[~is_valid_id, :])
```

```
Certains identifiants ne suivent pas le format attendu:
3019      T_0
5138      T_0
9668      T_0
```

```
10728     T_0
15292     T_0
          …
657830    T_0
662081    T_0
670680    T_0
671647    T_0
679180    T_0
Name: id_prod, Length: 200, dtype: object
Nombre de lignes avec un format non valide : 200
       id_prod                             date session_id client_id
3019       T_0  test_2021-03-01 02:30:02.237419        s_0      ct_0
5138       T_0  test_2021-03-01 02:30:02.237425        s_0      ct_0
9668       T_0  test_2021-03-01 02:30:02.237437        s_0      ct_1
10728      T_0  test_2021-03-01 02:30:02.237436        s_0      ct_0
15292      T_0  test_2021-03-01 02:30:02.237430        s_0      ct_0
…          …                                 …        …        …
657830     T_0  test_2021-03-01 02:30:02.237417        s_0      ct_0
662081     T_0  test_2021-03-01 02:30:02.237427        s_0      ct_1
670680     T_0  test_2021-03-01 02:30:02.237449        s_0      ct_1
671647     T_0  test_2021-03-01 02:30:02.237424        s_0      ct_1
679180     T_0  test_2021-03-01 02:30:02.237425        s_0      ct_1

[200 rows x 4 columns]
```

4.3.2 - Supprimer les lignes avec les ids invalides

[21]:
```
# Drop rows with invalid IDs
transactions_cleaned = transactions.drop(transactions.index[~is_valid_id])
print(f"DataFrame nettoyé sans les lignes invalides :")
transactions_cleaned.info()
print(transactions_cleaned)
```

```
DataFrame nettoyé sans les lignes invalides :
<class 'pandas.core.frame.DataFrame'>
Index: 679332 entries, 0 to 679531
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id_prod     679332 non-null  object
 1   date        679332 non-null  object
 2   session_id  679332 non-null  object
 3   client_id   679332 non-null  object
dtypes: object(4)
memory usage: 25.9+ MB
       id_prod                        date session_id client_id
0       0_1518  2022-05-20 13:21:29.043970   s_211425     c_103
1        1_251  2022-02-02 07:55:19.149409   s_158752    c_8534
2       0_1277  2022-06-18 15:44:33.155329   s_225667    c_6714
```

```
3         2_209  2021-06-24 04:19:29.835891    s_52962    c_6941
4        0_1509  2023-01-11 08:22:08.194479   s_325227    c_4232
…           …                           …          …          …
679527   0_1551  2022-01-15 13:05:06.246925   s_150195    c_8489
679528    1_639  2022-03-19 16:03:23.429229   s_181434    c_4370
679529   0_1425  2022-12-20 04:33:37.584749   s_314704     c_304
679530   0_1994  2021-07-16 20:36:35.350579    s_63204    c_2227
679531    1_523  2022-09-28 01:12:01.973763   s_274568    c_3873

[679332 rows x 4 columns]
```

4.3.2 - Vérification de la colonne date

[22]:
```
print("Date la plus ancienne:", transactions_cleaned['date'].min())
print("Date la plus récente:", transactions_cleaned['date'].max())
```

```
Date la plus ancienne: 2021-03-01 00:01:07.843138
Date la plus récente: 2023-02-28 23:58:30.792755
```

[23]:
```
# Définir le pattern pour vérifier le format date
pattern = r'^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{6}$'
# Vérifier si les valeurs de la colonne correspondent au pattern
is_valid_date_format = transactions_cleaned['date'].astype(str).str.
 ↪match(pattern, na=False)
invalid_dates = transactions_cleaned.loc[~is_valid_date_format, 'date']
print("Lignes avec des dates ne suivant pas le format attendu :")
print(transactions_cleaned.loc[~is_valid_date_format])
```

```
Lignes avec des dates ne suivant pas le format attendu :
Empty DataFrame
Columns: [id_prod, date, session_id, client_id]
Index: []
```

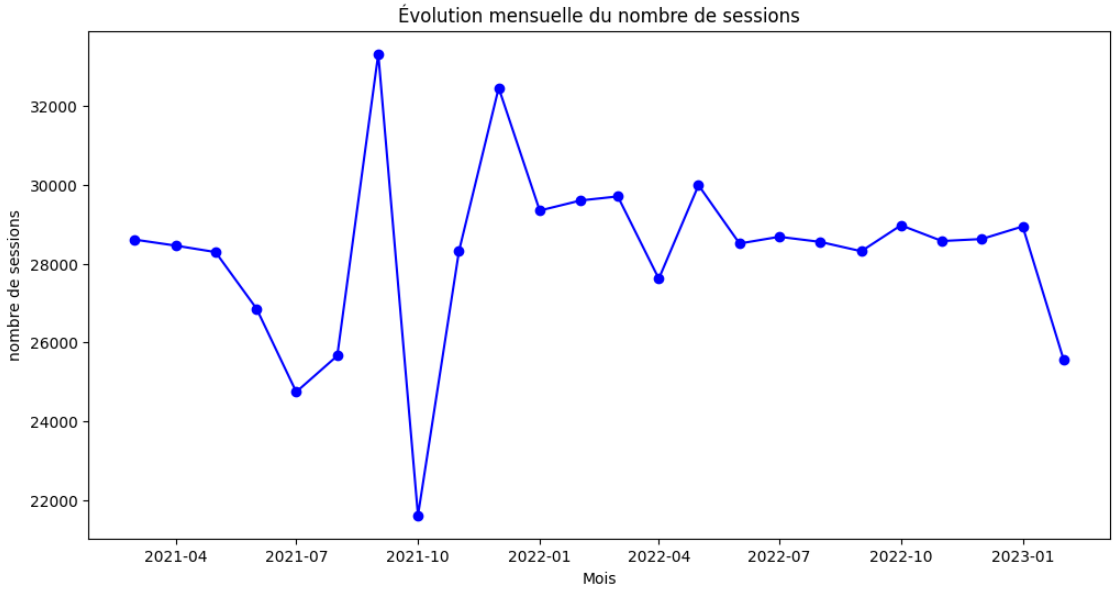
4.3 - Nombre de sessions et évolutions des sessions

[24]:
```
transactions_cleaned['date'] = pd.to_datetime(transactions['date'],␣
 ↪errors='coerce')
sessions_par_mois = transactions_cleaned.groupby(transactions_cleaned['date'].
 ↪dt.to_period('M'))['session_id'].count()
print("\nNombre de sessions par mois :\n", sessions_par_mois)
```

```
Nombre de sessions par mois :
 date
2021-03    28610
2021-04    28457
2021-05    28293
2021-06    26857
2021-07    24742
2021-08    25659
```

```
2021-09    33326
2021-10    21606
2021-11    28321
2021-12    32464
2022-01    29348
2022-02    29605
2022-03    29707
2022-04    27616
2022-05    29991
2022-06    28511
2022-07    28682
2022-08    28552
2022-09    28315
2022-10    28974
2022-11    28574
2022-12    28625
2023-01    28945
2023-02    25552
Freq: M, Name: session_id, dtype: int64
```

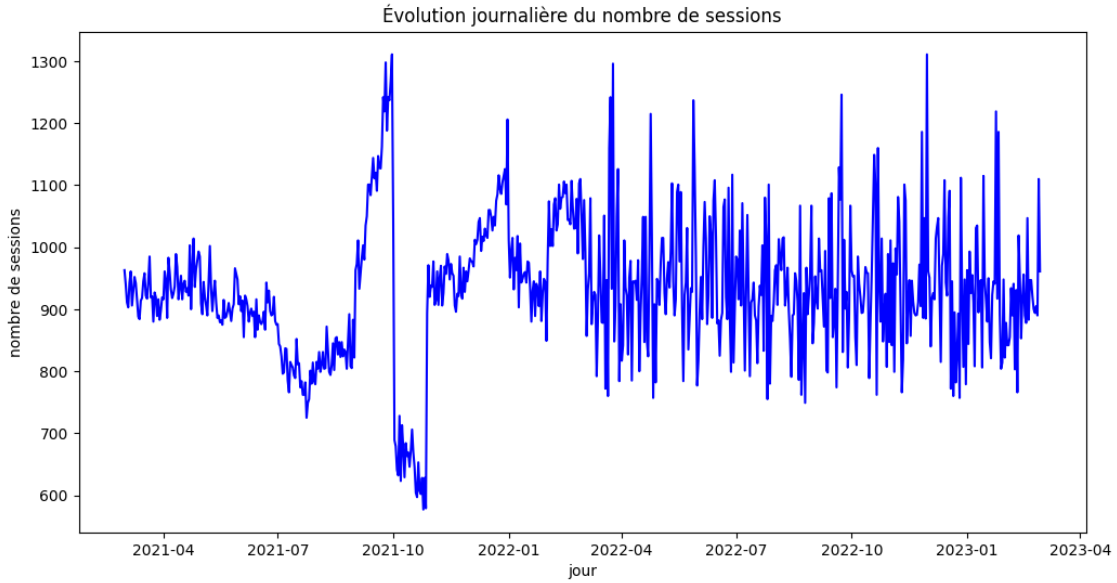
[25]:
```
sessions_par_mois.index = sessions_par_mois.index.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(sessions_par_mois.index, sessions_par_mois.values, marker='o',␣
 ↪linestyle='-', color='b')
plt.title('Évolution mensuelle du nombre de sessions')
plt.xlabel('Mois')
plt.ylabel('nombre de sessions')
plt.show()
```



[26]:
```
transactions_cleaned['date'] = pd.to_datetime(transactions['date'],␣
 ↪errors='coerce')
sessions_par_jour = transactions_cleaned.groupby(transactions_cleaned['date'].
 ↪dt.to_period('D'))['session_id'].count()
print("\nNombre de sessions par mois :\n", sessions_par_jour)
```

```
Nombre de sessions par mois :
 date
2021-03-01     963
2021-03-02     940
2021-03-03     911
2021-03-04     903
2021-03-05     943
              …
2023-02-24     894
2023-02-25     905
2023-02-26     890
2023-02-27    1110
2023-02-28     961
Freq: D, Name: session_id, Length: 730, dtype: int64
```

[27]:
```
sessions_par_jour.index = sessions_par_jour.index.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(sessions_par_jour.index, sessions_par_jour.values,color='b')
plt.title('Évolution journalière du nombre de sessions')
plt.xlabel('jour')
plt.ylabel('nombre de sessions')
plt.show()
```



4.4 - Sessions par catégories

4.4.1 - Récupération de de la catégorie

[28]:
```
categorie_session = transactions_cleaned.merge(products, on='id_prod', how=␣
↪'left')
categorie_session.info()
categorie_session.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 679332 entries, 0 to 679331
Data columns (total 6 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id_prod     679332 non-null  object
 1   date        679332 non-null  datetime64[ns]
 2   session_id  679332 non-null  object
 3   client_id   679332 non-null  object
 4   price       679111 non-null  float64
 5   categ       679111 non-null  float64
dtypes: datetime64[ns](1), float64(2), object(3)
memory usage: 31.1+ MB
```

[28]:

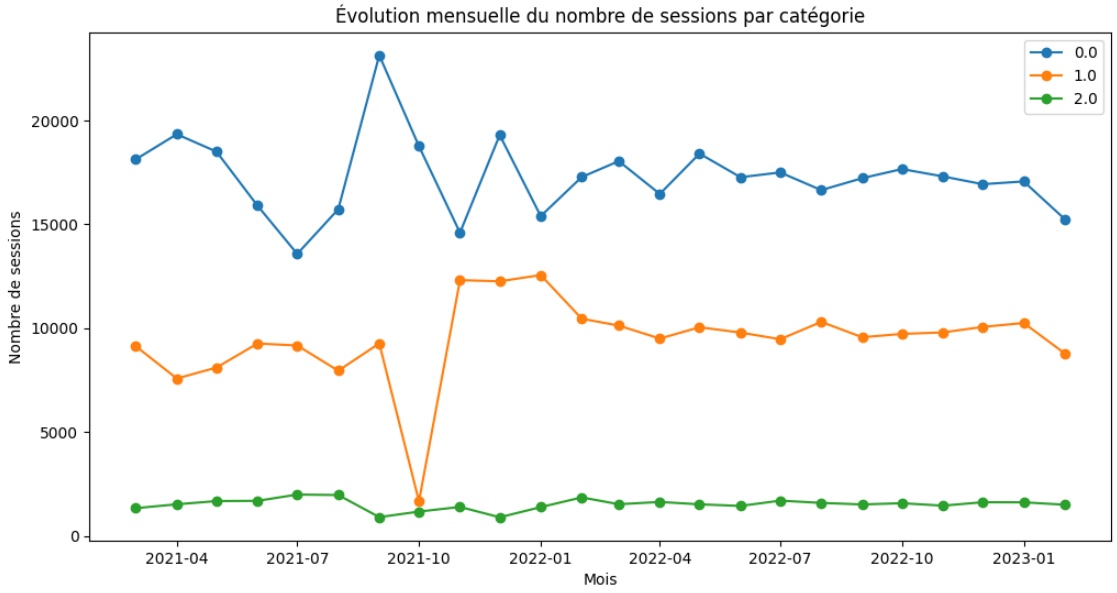
	id_prod	date	session_id	client_id	price	categ
0	0_1518	2022-05-20 13:21:29.043970	s_211425	c_103	4.18	0.0
1	1_251	2022-02-02 07:55:19.149409	s_158752	c_8534	15.99	1.0
2	0_1277	2022-06-18 15:44:33.155329	s_225667	c_6714	7.99	0.0
3	2_209	2021-06-24 04:19:29.835891	s_52962	c_6941	69.99	2.0
4	0_1509	2023-01-11 08:22:08.194479	s_325227	c_4232	4.99	0.0

4.4.2 -Evolution des session

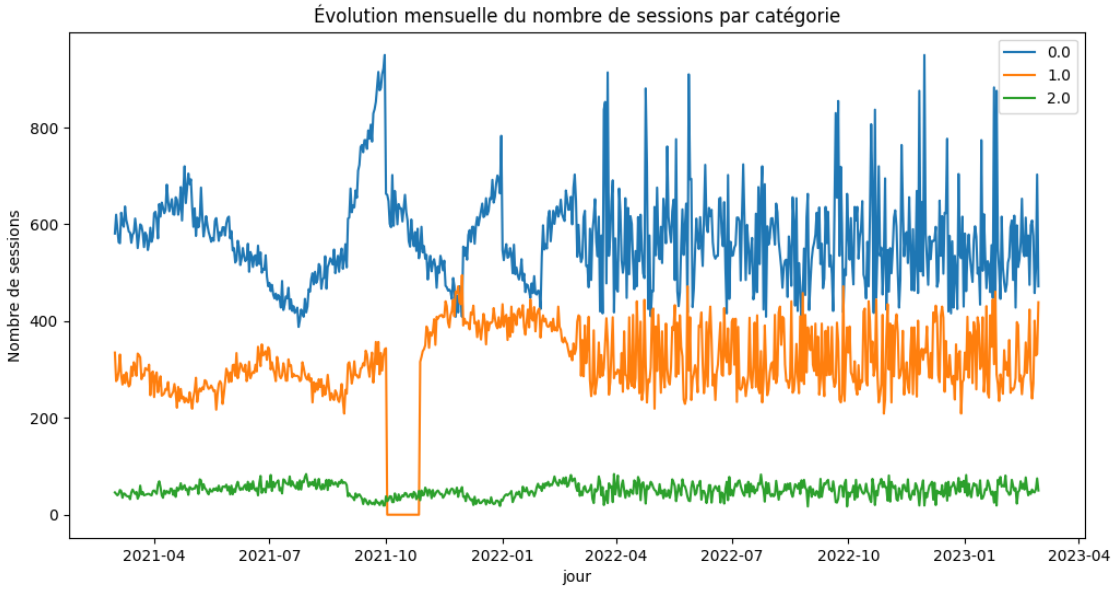
[29]:
```
#Évolution des sessions par catégorie et par mois
sessions_par_categorie_mois = categorie_session.groupby(['categ',␣
 ↪categorie_session['date'].dt.to_period('M')])['session_id'].count()

# Convertir la série en DataFrame et remplir les valeurs manquantes avec zéro
sessions_par_categorie_mois = sessions_par_categorie_mois.unstack(level=0).
 ↪fillna(0)

# Convertir l'index périodique en timestamp
sessions_par_categorie_mois.index = sessions_par_categorie_mois.index.
 ↪to_timestamp()

# Afficher le tableau
sessions_par_categorie_mois.head()
```

[29]:
```
categ         0.0   1.0   2.0
date
2021-03-01  18131  9134  1336
2021-04-01  19342  7579  1522
2021-05-01  18501  8107  1677
2021-06-01  15898  9264  1688
2021-07-01  13578  9169  1991
```

[30]:
```
sessions_par_categorie_mois = categorie_session.
 ↪pivot_table(index=categorie_session['date'].dt.to_period('M'),␣
 ↪columns='categ', values='session_id', aggfunc='count', fill_value=0)

# Convertir l'index périodique en timestamp
sessions_par_categorie_mois.index = sessions_par_categorie_mois.index.
 ↪to_timestamp()

plt.figure(figsize=(12, 6))

# Parcourir les catégories uniques
for categorie in sessions_par_categorie_mois.columns:
    plt.plot(sessions_par_categorie_mois.index,␣
 ↪sessions_par_categorie_mois[categorie], marker='o', linestyle='-',␣
 ↪label=str(categorie))

plt.title('Évolution mensuelle du nombre de sessions par catégorie')
plt.xlabel('Mois')
plt.ylabel('Nombre de sessions')
plt.legend()
plt.show()
```



[31]:
```
sessions_par_categorie_jour = categorie_session.
 ↪pivot_table(index=categorie_session['date'].dt.to_period('D'),␣
 ↪columns='categ', values='session_id', aggfunc='count', fill_value=0)

# Convertir l'index périodique en timestamp
sessions_par_categorie_jour.index = sessions_par_categorie_jour.index.
 ↪to_timestamp()

plt.figure(figsize=(12, 6))

# Parcourir les catégories uniques
for categorie in sessions_par_categorie_jour.columns:
    plt.plot(sessions_par_categorie_jour.index,␣
 ↪sessions_par_categorie_jour[categorie],label=str(categorie))

plt.title('Évolution mensuelle du nombre de sessions par catégorie')
plt.xlabel('jour')
plt.ylabel('Nombre de sessions')
plt.legend()
plt.show()
```




```
<h2 style="font-weight: bold; color: #001F3F;">Etape 5 - Merger les dataframes</h2>
```

5.1 - Importation des dataframes

[32]:
```
#Importation des fichiers
customers = pd.read_csv('/Users/Bouboule/Documents/Projet 6/BDD/customers.csv',␣
 ↪sep=';')
products = pd.read_csv('/Users/Bouboule/Documents/Projet 6/BDD/products.csv',␣
 ↪sep=';')
transactions = pd.read_csv('/Users/Bouboule/Documents/Projet 6/BDD/transactions.
 ↪csv', sep=';')
```

[33]:
```
customers.info()
customers.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
```

[33]:

	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
```

[34]:
```
products.info()
products.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
```

[34]:

	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

[35]:
```
transactions.info()
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
```

[35]:

	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

5.1 - Merge transactions & products

[36]:
```
# Fusion entre products et transactions
group1 = transactions.merge(products, on='id_prod', how= 'outer', indicator=␣
 ↪'true')
print(group1.shape)
group1.head()
```

```
(687555, 7)
```

[36]:

	id_prod	date	session_id	client_id	price	categ	true
0	0_1259	2021-03-01 00:01:07.843138	s_1	c_329	11.99	0	both
1	0_1259	2021-03-01 10:27:10.675023	s_202	c_1599	11.99	0	both
2	0_1259	2021-03-04 07:26:01.343183	s_1519	c_1609	11.99	0	both
3	0_1259	2021-03-05 13:32:43.907997	s_2114	c_107	11.99	0	both
4	0_1259	2021-03-07 09:47:05.272864	s_2976	c_4215	11.99	0	both

5.1.1 - Vérification de l'indicator

[37]:
```
# Compter le nombre de lignes où la valeur de la colonne 'true' est différente␣
 ↪de 'both'
verification_match = group1[group1['true'] != 'both']
print(verification_match.shape)
verification_match.info()
verification_match.head()
```

```
(21, 7)
<class 'pandas.core.frame.DataFrame'>
Index: 21 entries, 687534 to 687554
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id_prod     21 non-null     object
 1   date        0 non-null      object
 2   session_id  0 non-null      object
 3   client_id   0 non-null      object
 4   price       21 non-null     float64
 5   categ       21 non-null     int64
 6   true        21 non-null     category
dtypes: category(1), float64(1), int64(1), object(4)
memory usage: 1.3+ KB
```

[37]:

	id_prod	date	session_id	client_id	price	categ	true
687534	0_1016	NaN	NaN	NaN	35.06	0	right_only
687535	0_1780	NaN	NaN	NaN	1.67	0	right_only
687536	0_1062	NaN	NaN	NaN	20.08	0	right_only
687537	0_1119	NaN	NaN	NaN	2.99	0	right_only
687538	0_1014	NaN	NaN	NaN	1.15	0	right_only

[38]:
```
listeNan = group1.query('price.isna()')
print(group1.isna().sum())
```

```
id_prod        0
date          21
session_id    21
client_id     21
price          0
categ          0
true           0
dtype: int64
```

5.1.2 - Suppression des données non match

[39]:
```
group1 = group1[group1['true'] == 'both']
print(group1.shape)
group1.head()
```

```
(687534, 7)
```

[39]:

	id_prod	date	session_id	client_id	price	categ	true
0	0_1259	2021-03-01 00:01:07.843138	s_1	c_329	11.99	0	both
1	0_1259	2021-03-01 10:27:10.675023	s_202	c_1599	11.99	0	both
2	0_1259	2021-03-04 07:26:01.343183	s_1519	c_1609	11.99	0	both
3	0_1259	2021-03-05 13:32:43.907997	s_2114	c_107	11.99	0	both
4	0_1259	2021-03-07 09:47:05.272864	s_2976	c_4215	11.99	0	both

[40]:
```
group1 = group1.drop(['true'],axis=1)
group1.head()
```

[40]:

	id_prod	date	session_id	client_id	price	categ
0	0_1259	2021-03-01 00:01:07.843138	s_1	c_329	11.99	0
1	0_1259	2021-03-01 10:27:10.675023	s_202	c_1599	11.99	0
2	0_1259	2021-03-04 07:26:01.343183	s_1519	c_1609	11.99	0
3	0_1259	2021-03-05 13:32:43.907997	s_2114	c_107	11.99	0
4	0_1259	2021-03-07 09:47:05.272864	s_2976	c_4215	11.99	0

5.2 - Merge avec customers

5.2.1 - Merge

[41]:
```
# Fusion avec products
lapage = group1.merge(customers, on='client_id', how= 'outer',indicator= 'true'␣
↪)
print(lapage.shape)
lapage.head()
```

```
(687555, 9)
```

[41]:

	id_prod	date	session_id	client_id	price	categ	sex	\
0	0_1259	2021-03-01 00:01:07.843138	s_1	c_329	11.99	0.0	f	
1	0_1259	2022-10-01 00:01:07.843138	s_275943	c_329	11.99	0.0	f	
2	0_1259	2022-12-01 00:01:07.843138	s_305291	c_329	11.99	0.0	f	

```
3  0_1259  2023-01-01 00:01:07.843138   s_320153     c_329  11.99    0.0   f
4   1_397  2021-11-23 18:21:56.361813   s_123998     c_329  18.99    1.0   f

   birth  true
0   1967  both
1   1967  both
2   1967  both
3   1967  both
4   1967  both
```

5.2.2 - Vérification de l'indicator

[42]:
```
# Compter le nombre de lignes où la valeur de la colonne 'true' est différente de 'both'
verification_match = lapage[lapage['true'] != 'both']
print(verification_match.shape)
verification_match.info()
verification_match.head()
```

```
(21, 9)
<class 'pandas.core.frame.DataFrame'>
Index: 21 entries, 687534 to 687554
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id_prod     0 non-null      object
 1   date        0 non-null      object
 2   session_id  0 non-null      object
 3   client_id   21 non-null     object
 4   price       0 non-null      float64
 5   categ       0 non-null      float64
 6   sex         21 non-null     object
 7   birth       21 non-null     int64
 8   true        21 non-null     category
dtypes: category(1), float64(2), int64(1), object(5)
memory usage: 1.6+ KB
```

[42]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	true
687534	NaN	NaN	NaN	c_8253	NaN	NaN	f	2001	right_only
687535	NaN	NaN	NaN	c_3789	NaN	NaN	f	1997	right_only
687536	NaN	NaN	NaN	c_4406	NaN	NaN	f	1998	right_only
687537	NaN	NaN	NaN	c_2706	NaN	NaN	f	1967	right_only
687538	NaN	NaN	NaN	c_3443	NaN	NaN	m	1959	right_only

[43]:
```
listeNan = lapage.query('session_id.isna()')
print(group1.isna().sum())
```

```
id_prod       0
date          0
```

```
session_id    0
client_id     0
price         0
categ         0
dtype: int64
```

5.2.3 - Suppression des données pas match

[44]:
```
lapage = lapage[lapage['true'] == 'both']
print(lapage.shape)
lapage.head()
```

```
(687534, 9)
```

[44]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	true
0	0_1259	2021-03-01 00:01:07.843138	s_1	c_329	11.99	0.0	f	1967	both
1	0_1259	2022-10-01 00:01:07.843138	s_275943	c_329	11.99	0.0	f	1967	both
2	0_1259	2022-12-01 00:01:07.843138	s_305291	c_329	11.99	0.0	f	1967	both
3	0_1259	2023-01-01 00:01:07.843138	s_320153	c_329	11.99	0.0	f	1967	both
4	1_397	2021-11-23 18:21:56.361813	s_123998	c_329	18.99	1.0	f	1967	both

[45]:
```
lapage = lapage.drop(['true'],axis=1)
lapage.info()
lapage.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 687534 entries, 0 to 687533
Data columns (total 8 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
 4   price       687534 non-null  float64
 5   categ       687534 non-null  float64
 6   sex         687534 non-null  object
 7   birth       687534 non-null  int64
dtypes: float64(2), int64(1), object(5)
memory usage: 47.2+ MB
```

[45]:
```
  id_prod                       date session_id client_id  price  categ sex  \
0  0_1259  2021-03-01 00:01:07.843138        s_1     c_329  11.99    0.0   f
1  0_1259  2022-10-01 00:01:07.843138   s_275943     c_329  11.99    0.0   f
2  0_1259  2022-12-01 00:01:07.843138   s_305291     c_329  11.99    0.0   f
3  0_1259  2023-01-01 00:01:07.843138   s_320153     c_329  11.99    0.0   f
4   1_397  2021-11-23 18:21:56.361813   s_123998     c_329  18.99    1.0   f

   birth
0   1967
1   1967
2   1967
3   1967
4   1967
```

```
<h2 style="font-weight: bold; color: #001F3F;">Etape 6 - analyse des indicateurs de vente : Le
```

6.1 - Chiffre d'affaires

6.1.1 - Chiffre d'affaires par produit

[46]:
```
revenue_per_category = lapage.groupby('categ')['price'].sum()
print("\nChiffre d'affaires par catégorie de produit :\n", revenue_per_category)
```

```
Chiffre d'affaires par catégorie de produit :
 categ
0.0    4419730.97
1.0    4827657.11
2.0    2780275.02
Name: price, dtype: float64
```

6.1.2 - Chiffre d'affaires par client

[47]:
```
revenue_per_customer = lapage.groupby('client_id')['price'].sum()
print("\nChiffre d'affaires par client :\n", revenue_per_customer)
```

```
Chiffre d'affaires par client :
 client_id
c_1        629.02
c_10      1353.60
c_100      254.85
c_1000    2291.88
c_1001    1823.85
           …
c_995      189.41
c_996     1637.34
c_997     1490.01
c_998     2822.22
c_999      701.40
```

```
Name: price, Length: 8600, dtype: float64
```

6.1.1 - Chiffre d'affaires total

[48]:
```
total_revenue = lapage['price'].sum()
print(f"Chiffre d'affaires total : {total_revenue:.2f}")
```

```
Chiffre d'affaires total : 12027663.10
```

Chiffre d'affaire par jour

[49]:
```
# Convertir la colonne 'date' en type datetime
lapage['date'] = pd.to_datetime(lapage['date'], errors='coerce')

# Créer un DataFrame regroupé par jour avec une colonne pour le jour
chiffre_affaires_par_jour = lapage.groupby(lapage['date'].dt.to_period("D")).
 ↪agg({'price': 'sum'}).reset_index()
# Afficher les résultats
print("Chiffre d'affaires par jour avec colonne jour :\n",␣
 ↪chiffre_affaires_par_jour)
```

```
Chiffre d'affaires par jour avec colonne jour :
           date     price
0    2021-03-01  16565.22
1    2021-03-02  15486.45
2    2021-03-03  15198.69
3    2021-03-04  15196.07
4    2021-03-05  17471.37
..          …       …
725  2023-02-24  15207.89
726  2023-02-25  15761.25
727  2023-02-26  16304.72
728  2023-02-27  19170.81
729  2023-02-28  18105.15

[730 rows x 2 columns]
```

Chiffre d'affaire par mois

[50]:
```
# Créer un DataFrame regroupé par mois avec une colonne pour la période (mois)
chiffre_affaires_par_mois = lapage.groupby(lapage['date'].dt.to_period("M")).
 ↪agg({'price': 'sum'}).reset_index()
# Afficher les résultats
print("\nChiffre d'affaires par mois avec colonne mois :\n",␣
 ↪chiffre_affaires_par_mois)
```

```
Chiffre d'affaires par mois avec colonne mois :
       date      price
0   2021-03  482440.61
1   2021-04  476109.30
```

```
2   2021-05  492943.47
3   2021-06  484088.56
4   2021-07  482835.40
5   2021-08  482284.79
6   2021-09  507240.68
7   2021-10  489743.61
8   2021-11  516167.73
9   2021-12  525917.28
10  2022-01  525338.99
11  2022-02  535571.50
12  2022-03  515456.53
13  2022-04  492998.94
14  2022-05  517132.60
15  2022-06  496016.12
16  2022-07  510783.12
17  2022-08  506467.27
18  2022-09  494114.53
19  2022-10  507917.77
20  2022-11  496664.94
21  2022-12  510219.50
22  2023-01  517540.55
23  2023-02  456679.76
```

Chiffre d'affaire par an

[51]:
```
chiffre_affaires_par_an = lapage.groupby(lapage['date'].dt.to_period("Y")).
 ↪agg({'price': 'sum'}).reset_index()
# Afficher les résultats
print("\nChiffre d'affaires par an avec colonne année :\n",␣
 ↪chiffre_affaires_par_an)
```

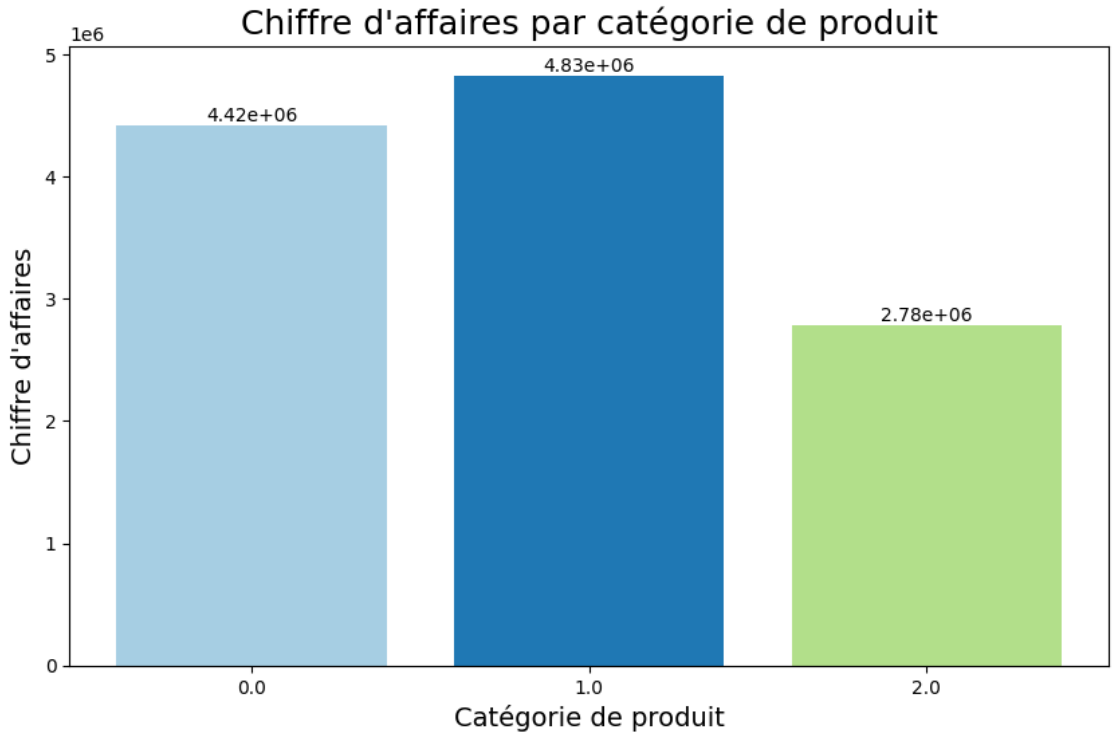
```
Chiffre d'affaires par an avec colonne année :
    date       price
0  2021  4939771.43
1  2022  6108681.81
2  2023   974220.31
```

6.2 - Graphiques

6.2.1 - Graphique à barres du chiffre d'affaires par catégorie

[52]:
```
# Convertir les indices de float à des chaînes
categories_str = revenue_per_category.index.astype(str)
revenue_per_category_values = revenue_per_category.values

plt.figure(figsize=(10, 6))
bars = plt.bar(categories_str, revenue_per_category_values, color=plt.cm.Paired.
 ↪colors)
```

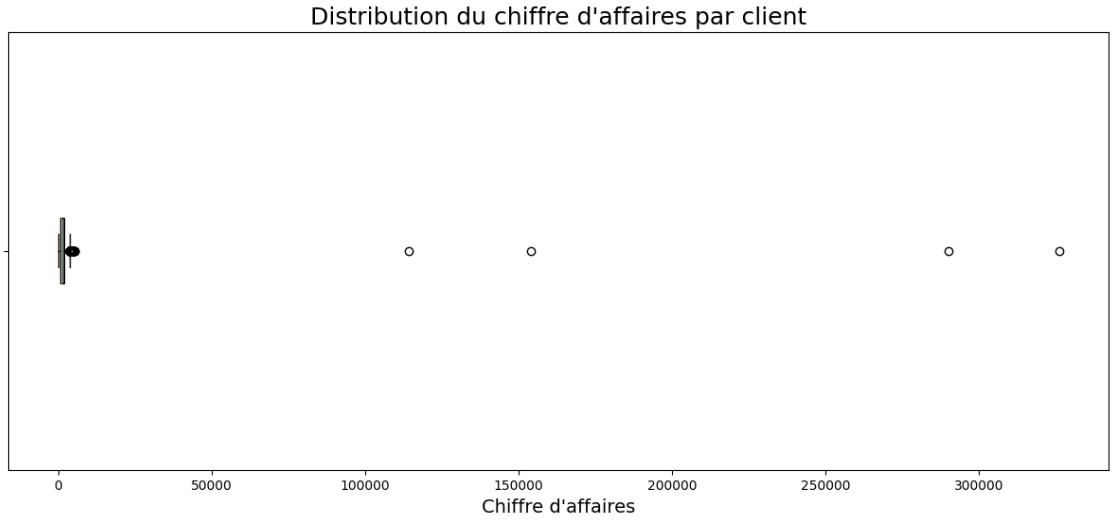
```
# Ajout des valeurs sur les barres
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.2e}', ha='center',␣
 ↪va='bottom')

plt.title('Chiffre d\'affaires par catégorie de produit', fontsize=18)
plt.xlabel('Catégorie de produit', fontsize=14)
plt.ylabel('Chiffre d\'affaires', fontsize=14)

plt.savefig('chiffre_affaire_categorie.png', transparent=True, dpi=300)
plt.show()
```



6.2.2 - Distribution du chiffre d'affaire par client

[53]:
```
plt.figure(figsize=(15, 6))

box = plt.boxplot(revenue_per_customer.values, vert=False, patch_artist=True)

# Personnalisation des couleurs du boxplot
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']
for patch, color in zip(box['boxes'], colors):
```

```
    patch.set_facecolor(color)

plt.yticks([1], [""])  # Supprimer les étiquettes de l'axe y
plt.title('Distribution du chiffre d\'affaires par client', fontsize=18)
plt.xlabel('Chiffre d\'affaires', fontsize=14)


plt.savefig('boxplot_chiffreaffaire_client.png', transparent=True, dpi=300)

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

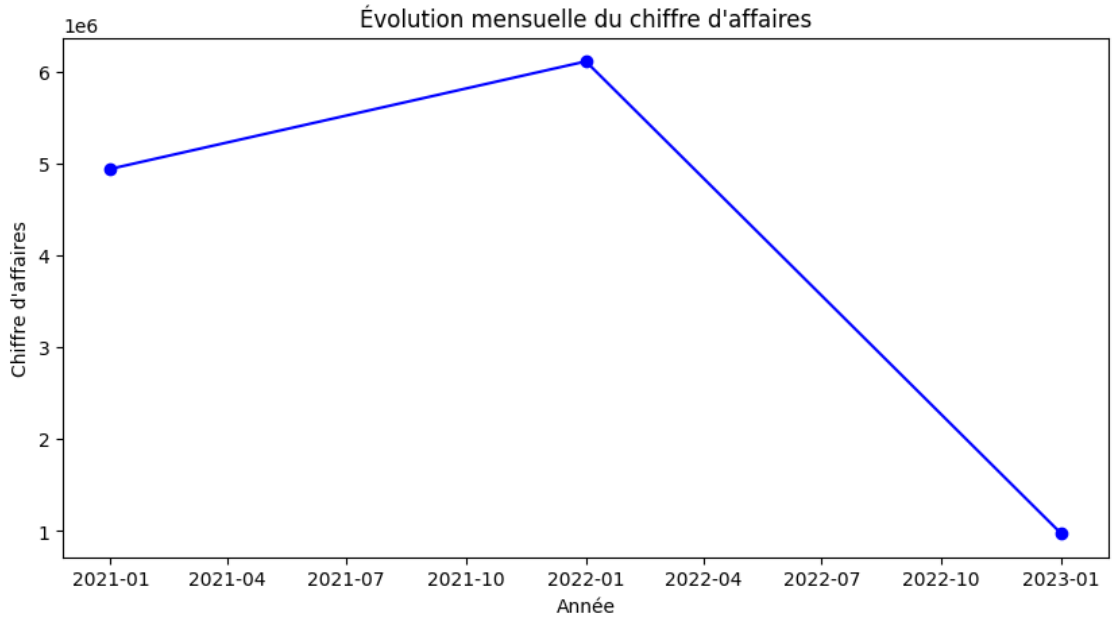


6.2.3 - Evolution du chiffre d'affaires

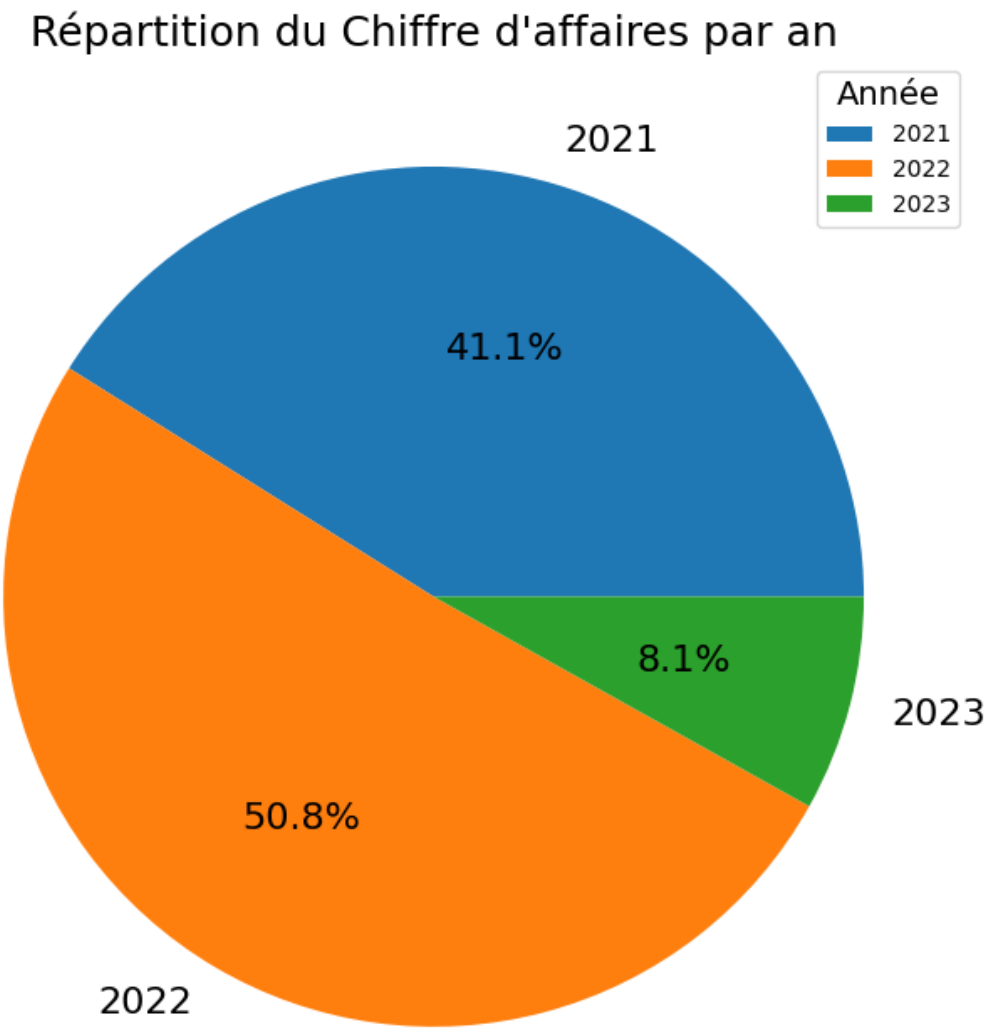
Chiffre d'affaire par an

[54]:
```
# Convertir la colonne 'price' en valeurs numériques
chiffre_affaires_par_an['price'] = chiffre_affaires_par_an['price'].
 ↪astype(float)

# Extraire la valeur entière de la période dans la colonne 'date'
chiffre_affaires_par_an['date'] = chiffre_affaires_par_an['date'].dt.
 ↪to_timestamp()


# Tracer le graphique
plt.figure(figsize=(10, 5))
plt.plot(chiffre_affaires_par_an['date'], chiffre_affaires_par_an['price'],␣
 ↪marker='o', linestyle='-', color='b')
plt.title('Évolution mensuelle du chiffre d\'affaires')
plt.xlabel('Année')
```

```
plt.ylabel('Chiffre d\'affaires')
plt.show()
```



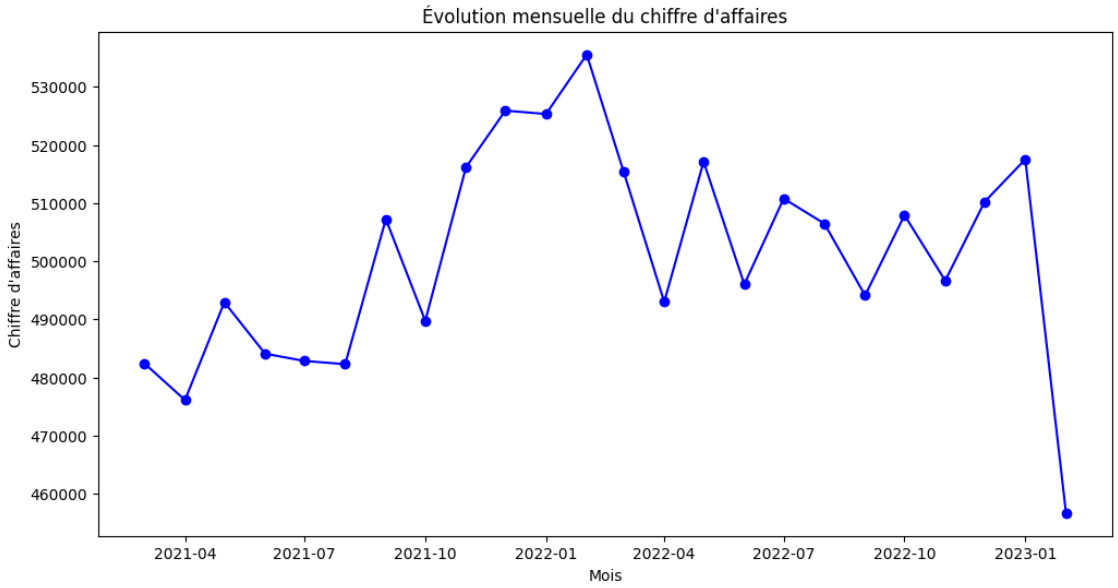
[55]:
```
plt.figure(figsize=(7, 7))
plt.pie(chiffre_affaires_par_an['price'],␣
 ↪labels=chiffre_affaires_par_an['date'].dt.year,
        autopct=lambda p: '{:.1f}%'.format(p) if p > 0 else '',
        pctdistance=0.6,
        textprops={'fontsize': 16}),

plt.legend(loc="best", fontsize=10, title="Année", title_fontsize='14')

plt.title('Répartition du Chiffre d\'affaires par an', fontsize=18)
plt.tight_layout()

# Sauvegarde du graphique avec un fond transparent
plt.savefig('chiffre_affaire_camembert.png', transparent=True)
plt.show()
```



Chiffre d'affaire par mois

[56]:
```
# Convertir la colonne 'price' en valeurs numériques
chiffre_affaires_par_mois['price'] = chiffre_affaires_par_mois['price'].
 ↪astype(float)

# Extraire la valeur entière de la période dans la colonne 'date'
chiffre_affaires_par_mois['date'] = chiffre_affaires_par_mois['date'].dt.
 ↪to_timestamp()

# Tracer le graphique
plt.figure(figsize=(12, 6))
```

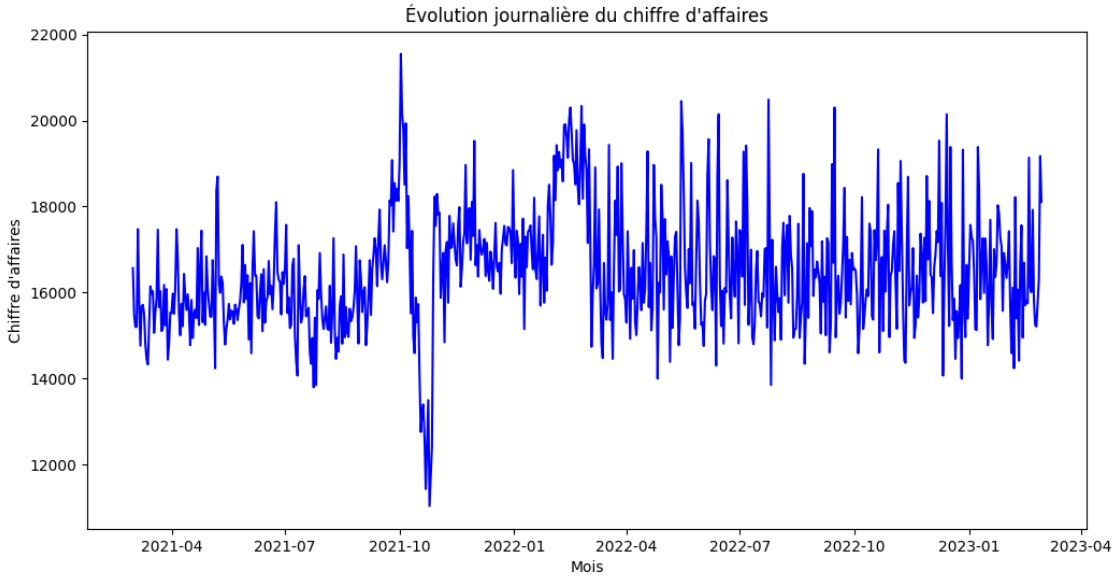
```
plt.plot(chiffre_affaires_par_mois['date'], chiffre_affaires_par_mois['price'],␣
 ↪marker='o', linestyle='-', color='b')
plt.title('Évolution mensuelle du chiffre d\'affaires')
plt.xlabel('Mois')
plt.ylabel('Chiffre d\'affaires')
plt.savefig('evolution_chiffre_affaire_mois.png', transparent=True)
plt.show()
```



Chiffre d'affaire par jour

[57]:
```
# Convertir la colonne 'price' en valeurs numériques
chiffre_affaires_par_jour['price'] = chiffre_affaires_par_jour['price'].
 ↪astype(float)

# Extraire la valeur entière de la période dans la colonne 'date'
chiffre_affaires_par_jour['date'] = chiffre_affaires_par_jour['date'].dt.
 ↪to_timestamp()
```

[58]:
```
# Tracer le graphique
plt.figure(figsize=(12, 6))
plt.plot(chiffre_affaires_par_jour['date'], chiffre_affaires_par_jour['price'],␣
 ↪color='b')
plt.title('Évolution journalière du chiffre d\'affaires')
plt.xlabel('Mois')
plt.ylabel('Chiffre d\'affaires')
plt.savefig('evolution_chiffre_affaire_jour.png', transparent=True)
plt.show()
```



6.4 - Evolution du nombre de session

6.5.1 - Evolution du nombre de session toutes catégories

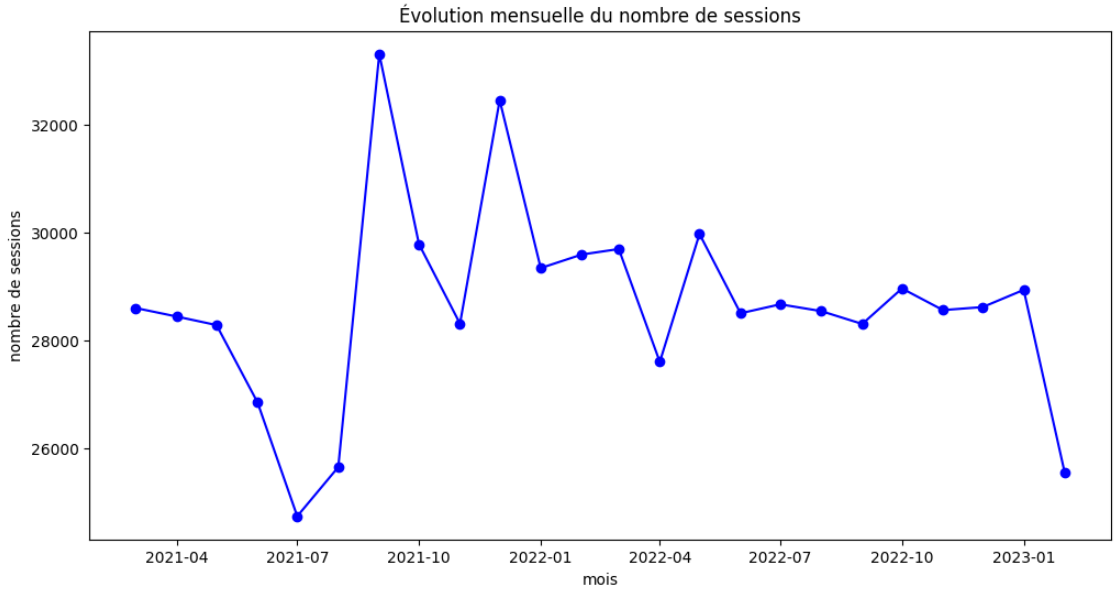
Evolution par mois

[59]:
```
total_session = lapage['session_id'].count()
print(f"Nombre de session : {total_session:.2f}")

lapage['date'] = pd.to_datetime(lapage['date'], errors='coerce')
sessions_par_mois = lapage.groupby(lapage['date'].dt.
 ↪to_period('M'))['session_id'].count()
print("\nNombre de sessions par mois :\n", sessions_par_mois)
```

```
Nombre de session : 687534.00

Nombre de sessions par mois :
 date
2021-03    28601
2021-04    28443
2021-05    28285
2021-06    26850
2021-07    24738
2021-08    25650
2021-09    33314
2021-10    29786
2021-11    28311
2021-12    32457
2022-01    29343
```

```
2022-02    29594
2022-03    29696
2022-04    27602
2022-05    29975
2022-06    28504
2022-07    28670
2022-08    28544
2022-09    28306
2022-10    28964
2022-11    28563
2022-12    28619
2023-01    28938
2023-02    25545
Freq: M, Name: session_id, dtype: int64
```

[60]:
```
sessions_par_mois.index = sessions_par_mois.index.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(sessions_par_mois.index, sessions_par_mois.values,marker='o',␣
 ↪linestyle='-',color='b')
plt.title('Évolution mensuelle du nombre de sessions')
plt.xlabel('mois')
plt.ylabel('nombre de sessions')
plt.savefig('evolution_session_mois.png', transparent=True)
plt.show()
```

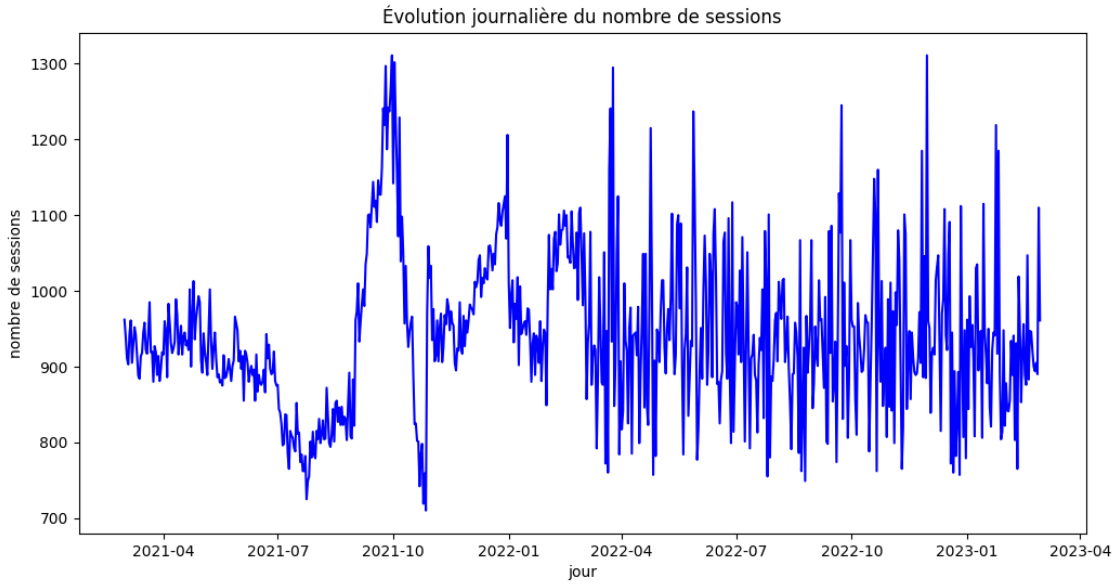


Evolution par jour

[61]:
```
lapage['date'] = pd.to_datetime(lapage['date'], errors='coerce')
sessions_par_jour = lapage.groupby(lapage['date'].dt.
 ↪to_period('D'))['session_id'].count()
print("\nNombre de sessions par jour :\n", sessions_par_jour)
```

```
Nombre de sessions par jour :
 date
2021-03-01     962
2021-03-02     939
2021-03-03     911
2021-03-04     903
2021-03-05     943
              …
2023-02-24     894
2023-02-25     905
2023-02-26     890
2023-02-27    1110
2023-02-28     961
Freq: D, Name: session_id, Length: 730, dtype: int64
```

[62]:
```
sessions_par_jour.index = sessions_par_jour.index.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(sessions_par_jour.index, sessions_par_jour.values,color='b')
plt.title('Évolution journalière du nombre de sessions')
plt.xlabel('jour')
plt.ylabel('nombre de sessions')
plt.savefig('evolution_session_jour.png', transparent=True)
plt.show()
```



6.5.2 - Evolution du nombre de session par catégorie

Evolution par mois

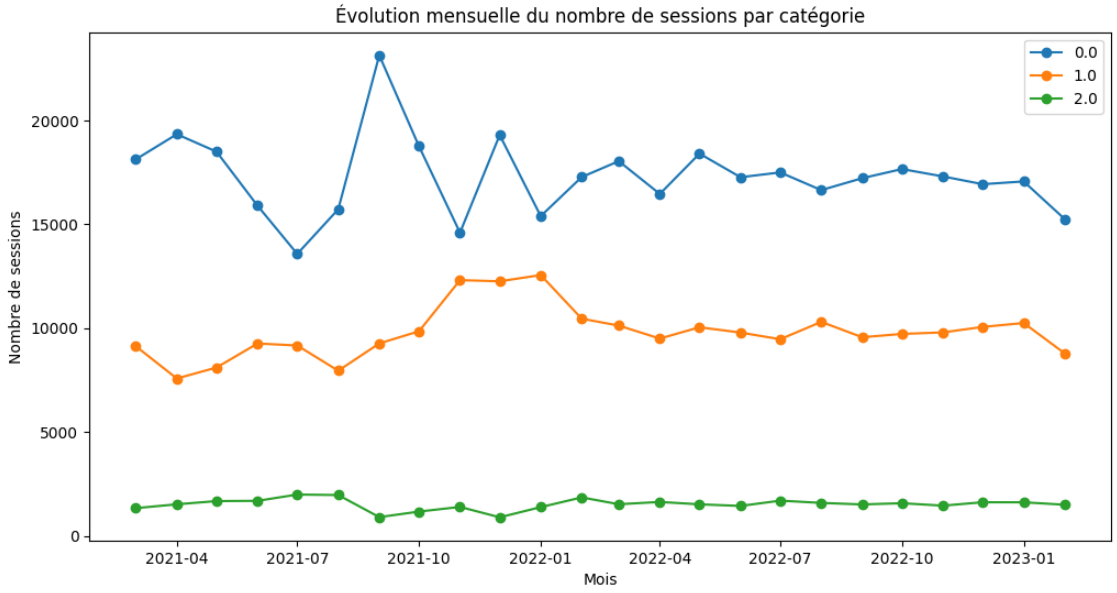
[63]:
```
# Évolution des sessions par catégorie et mois
sessions_par_categorie_mois = lapage.groupby(['categ', lapage['date'].dt.
 ↪to_period('M')])['session_id'].count()

# Reset the index to convert it to a DataFrame
sessions_par_categorie_mois = sessions_par_categorie_mois.reset_index()

# Convert the DateTime part of the MultiIndex to Timestamp
sessions_par_categorie_mois['date'] = sessions_par_categorie_mois['date'].dt.
 ↪to_timestamp()

# Afficher le tableau
sessions_par_categorie_mois.head()
```

[63]:

	categ	date	session_id
0	0.0	2021-03-01	18131
1	0.0	2021-04-01	19342
2	0.0	2021-05-01	18501
3	0.0	2021-06-01	15898
4	0.0	2021-07-01	13578

[64]:
```
sessions_par_categorie_mois = lapage.pivot_table(index=lapage['date'].dt.
 ↪to_period('M'), columns='categ', values='session_id', aggfunc='count',␣
 ↪fill_value=0)

# Convertir l'index périodique en timestamp
sessions_par_categorie_mois.index = sessions_par_categorie_mois.index.
 ↪to_timestamp()

plt.figure(figsize=(12, 6))

# Parcourir les catégories uniques
for categorie in sessions_par_categorie_mois.columns:
    plt.plot(sessions_par_categorie_mois.index,␣
 ↪sessions_par_categorie_mois[categorie], marker='o', linestyle='-',␣
 ↪label=str(categorie))

plt.title('Évolution mensuelle du nombre de sessions par catégorie')
plt.xlabel('Mois')
plt.ylabel('Nombre de sessions')
plt.legend()
```

```
plt.savefig('evolution_session_categorie_mois.png', transparent=True)
plt.show()
```

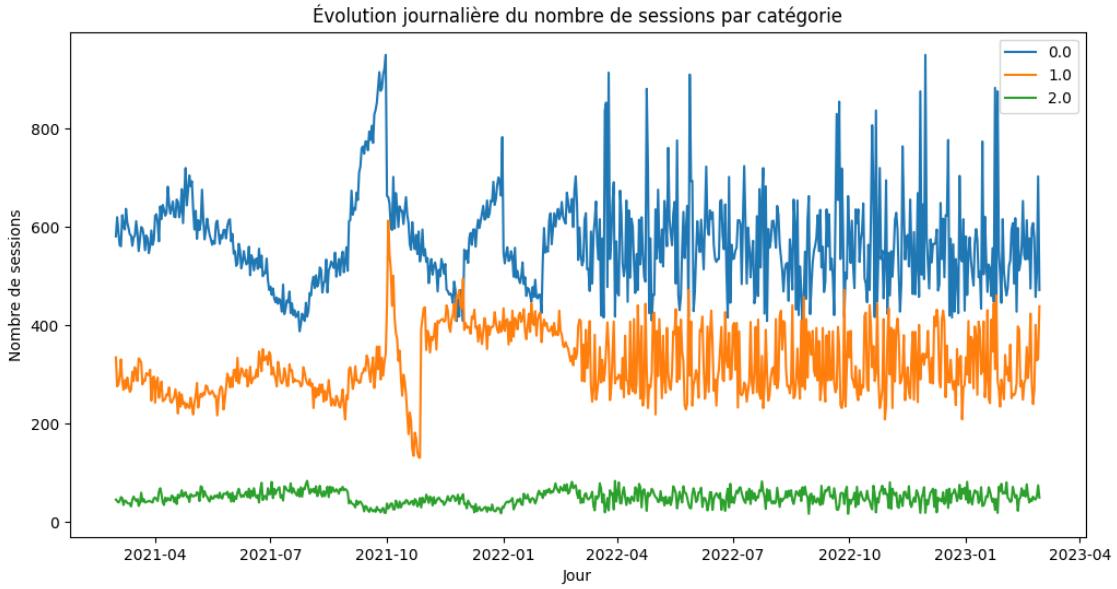


Evolution par jour

[65]:
```
#Évolution des sessions par catégorie et par mois
sessions_par_categorie_jour = lapage.groupby(['categ', lapage['date'].dt.
 ↪to_period('D')])['session_id'].count()

# Convertir la série en DataFrame et remplir les valeurs manquantes avec zéro
sessions_par_categorie_jour = sessions_par_categorie_jour.unstack(level=0).
 ↪fillna(0)

# Convertir l'index périodique en timestamp
sessions_par_categorie_jour.index = sessions_par_categorie_jour.index.
 ↪to_timestamp()

# Afficher le tableau
sessions_par_categorie_jour.head()
```

[65]:
```
categ       0.0  1.0  2.0
date
2021-03-01  581  335   46
2021-03-02  620  276   43
2021-03-03  591  280   40
2021-03-04  563  297   43
2021-03-05  561  331   51
```

[66]:
```
sessions_par_categorie_jour = lapage.pivot_table(index=lapage['date'].dt.
 ↪to_period('D'), columns='categ', values='session_id', aggfunc='count',␣
 ↪fill_value=0)

# Convertir l'index périodique en timestamp
sessions_par_categorie_jour.index = sessions_par_categorie_jour.index.
 ↪to_timestamp()

plt.figure(figsize=(12, 6))

# Parcourir les catégories uniques
for categorie in sessions_par_categorie_jour.columns:
    plt.plot(sessions_par_categorie_jour.index,␣
 ↪sessions_par_categorie_jour[categorie],label=str(categorie))

plt.title('Évolution journalière du nombre de sessions par catégorie')
plt.xlabel('Jour')
plt.ylabel('Nombre de sessions')
plt.legend()
plt.savefig('evolution_session_categorie_jour.png', transparent=True)
plt.show()
```



6.5 - Evolution du chiffre d'affaire par catégorie

6.5.1 - Evolution du chiffre d'affaires par catégorie

Chiffre d'affaires par catégorie par jour

[67]:
```
# Chiffre d'affaires par catégorie par jour
chiffre_affaires_categorie_par_jour = lapage.groupby(['categ', lapage['date'].
 ↪dt.to_period("D")])['price'].sum().reset_index()

# Afficher les résultats
print("Chiffre d'affaires par catégorie par jour :\n",␣
 ↪chiffre_affaires_categorie_par_jour)
```

```
Chiffre d'affaires par catégorie par jour :
       categ        date    price
0        0.0  2021-03-01  6262.65
1        0.0  2021-03-02  6718.27
2        0.0  2021-03-03  6121.03
3        0.0  2021-03-04  5891.48
4        0.0  2021-03-05  5975.97
…      …          …       …
2185     2.0  2023-02-24  3538.90
2186     2.0  2023-02-25  2767.95
2187     2.0  2023-02-26  3772.73
2188     2.0  2023-02-27  5271.62
2189     2.0  2023-02-28  3977.04

[2190 rows x 3 columns]
```

Chiffre d'affaires par catégorie par mois

[68]:
```
# Chiffre d'affaires par catégorie par mois
chiffre_affaires_categorie_par_mois = lapage.groupby(['categ', lapage['date'].
 ↪dt.to_period("M")])['price'].sum().reset_index()

# Afficher les résultats
print("Chiffre d'affaires par catégorie par mois :\n",␣
 ↪chiffre_affaires_categorie_par_mois)
```

```
Chiffre d'affaires par catégorie par mois :
     categ     date      price
0      0.0  2021-03  193629.17
1      0.0  2021-04  205222.46
2      0.0  2021-05  196186.72
3      0.0  2021-06  167943.15
4      0.0  2021-07  144750.79
..     …       …         …
67     2.0  2022-10  120878.94
68     2.0  2022-11  111642.60
69     2.0  2022-12  123803.09
70     2.0  2023-01  126153.08
71     2.0  2023-02  113875.52

[72 rows x 3 columns]
```

Chiffre d'affaires par catégorie par an

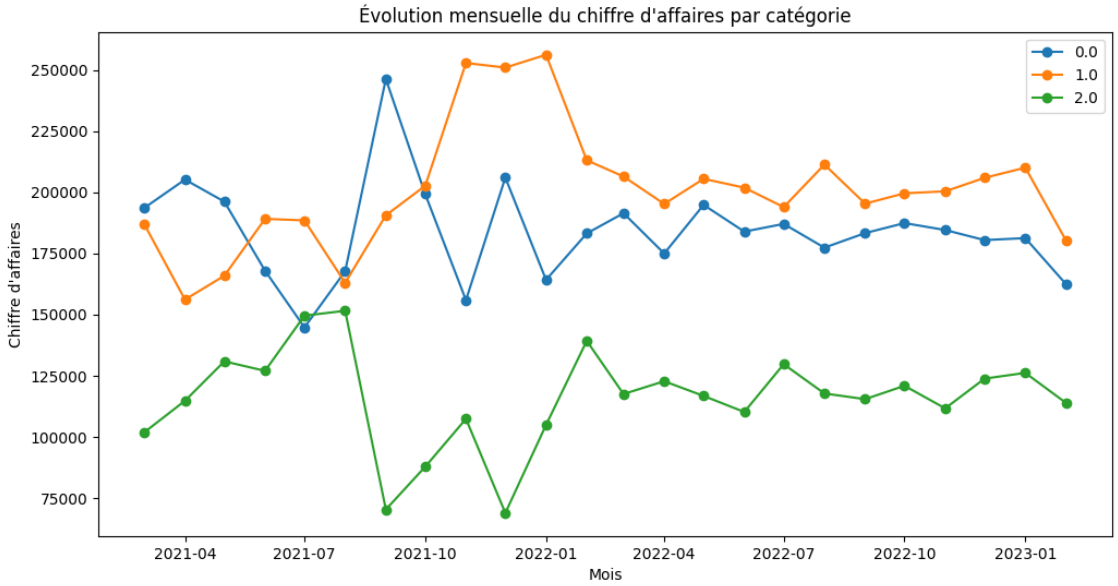
[69]:
```
# Chiffre d'affaires par catégorie par an
chiffre_affaires_categorie_par_an = lapage.groupby(['categ', lapage['date'].dt.
 ↪to_period("Y")])['price'].sum().reset_index()

# Afficher les résultats
print("Chiffre d'affaires par catégorie par an :\n",␣
 ↪chiffre_affaires_categorie_par_an)
```

```
Chiffre d'affaires par catégorie par an :
    categ  date       price
0     0.0  2021  1883020.45
1     0.0  2022  2192970.46
2     0.0  2023   343740.06
3     1.0  2021  1946940.72
4     1.0  2022  2485275.19
5     1.0  2023   390451.65
6     2.0  2021  1109810.26
7     2.0  2022  1430436.16
8     2.0  2023   240028.60
```

6.5.2 - Graphique d'évolution du chiffre d'affaires par catégorie

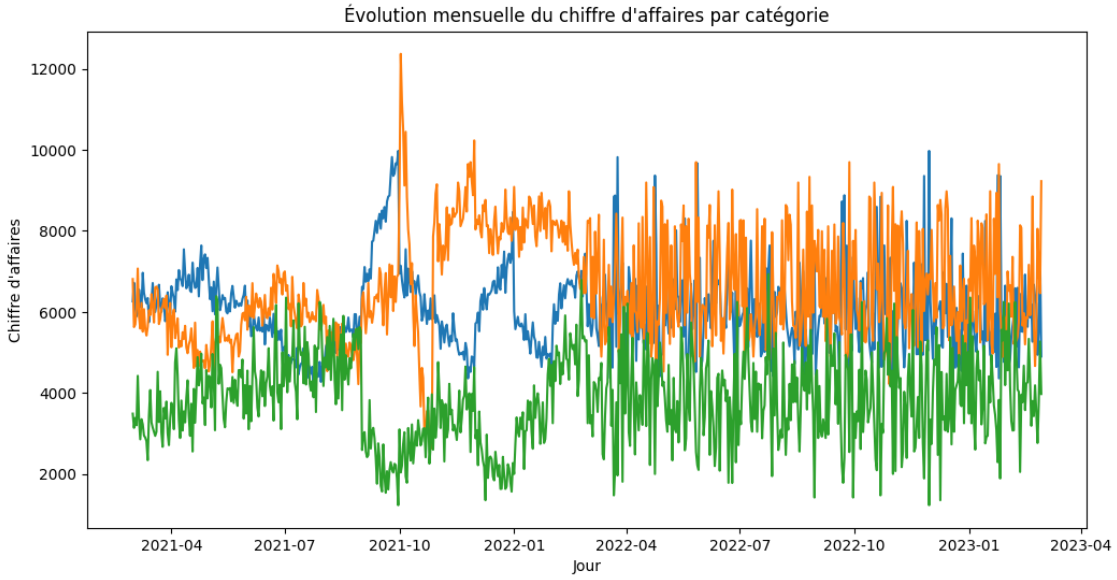
[70]:
```
# Convertir l'index multi-niveau en colonnes
chiffre_affaires_categorie_par_mois = chiffre_affaires_categorie_par_mois.
 ↪reset_index()

# Convertir la colonne 'date' de type période à type timestamp
chiffre_affaires_categorie_par_mois['date'] =␣
 ↪chiffre_affaires_categorie_par_mois['date'].dt.to_timestamp()

# Définissez l'index sur la colonne 'date'
chiffre_affaires_categorie_par_mois.set_index('date', inplace=True)

plt.figure(figsize=(12, 6))

# Parcourez les catégories uniques
for categorie in chiffre_affaires_categorie_par_mois['categ'].unique():
    data_category =␣
 ↪chiffre_affaires_categorie_par_mois[chiffre_affaires_categorie_par_mois['categ']␣
 ↪== categorie]
    plt.plot(data_category.index, data_category['price'], marker='o',␣
 ↪linestyle='-', label=categorie)

plt.title('Évolution mensuelle du chiffre d\'affaires par catégorie')
plt.xlabel('Mois')
plt.ylabel('Chiffre d\'affaires')
```

```
plt.legend()
plt.savefig('evolution_chiffre_affaire_categorie_mois.png', transparent=True)
plt.show()
```



[71]:
```
# Convertir l'index multi-niveau en colonnes
chiffre_affaires_categorie_par_jour = chiffre_affaires_categorie_par_jour.
 ↪reset_index()

# Convertir la colonne 'date' de type période à type timestamp
chiffre_affaires_categorie_par_jour['date'] =␣
 ↪chiffre_affaires_categorie_par_jour['date'].dt.to_timestamp()

# Définissez l'index sur la colonne 'date'
chiffre_affaires_categorie_par_jour.set_index('date', inplace=True)

plt.figure(figsize=(12, 6))

# Parcourez les catégories uniques
for categorie in chiffre_affaires_categorie_par_jour['categ'].unique():
    data_category =␣
 ↪chiffre_affaires_categorie_par_jour[chiffre_affaires_categorie_par_jour['categ']␣
 ↪== categorie]
    plt.plot(data_category.index, data_category['price'], label=categorie)

plt.title('Évolution mensuelle du chiffre d\'affaires par catégorie')
plt.xlabel('Jour')
plt.ylabel('Chiffre d\'affaires')
```

```
plt.savefig('evolution_chiffre_affaire_categorie_jour.png', transparent=True)
plt.show()
```



<h2 style="font-weight: bold; color: #001F3F;">Etape 7 - analyse des indicateurs de vente : Evo

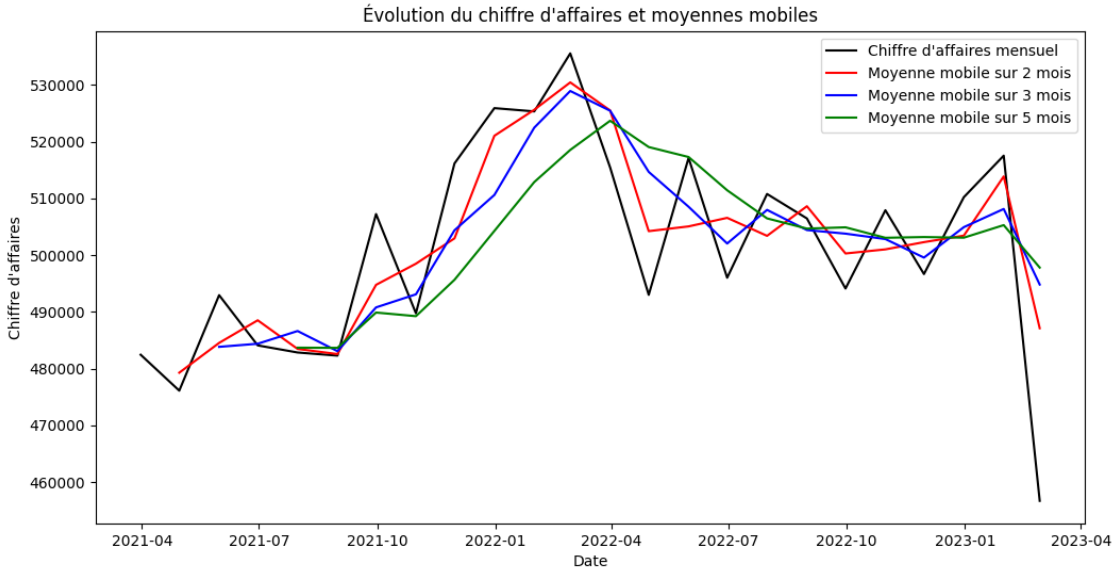
7.1 - Moyenne mobile

7.1.1 - Chiffre d'affaire

Par mois

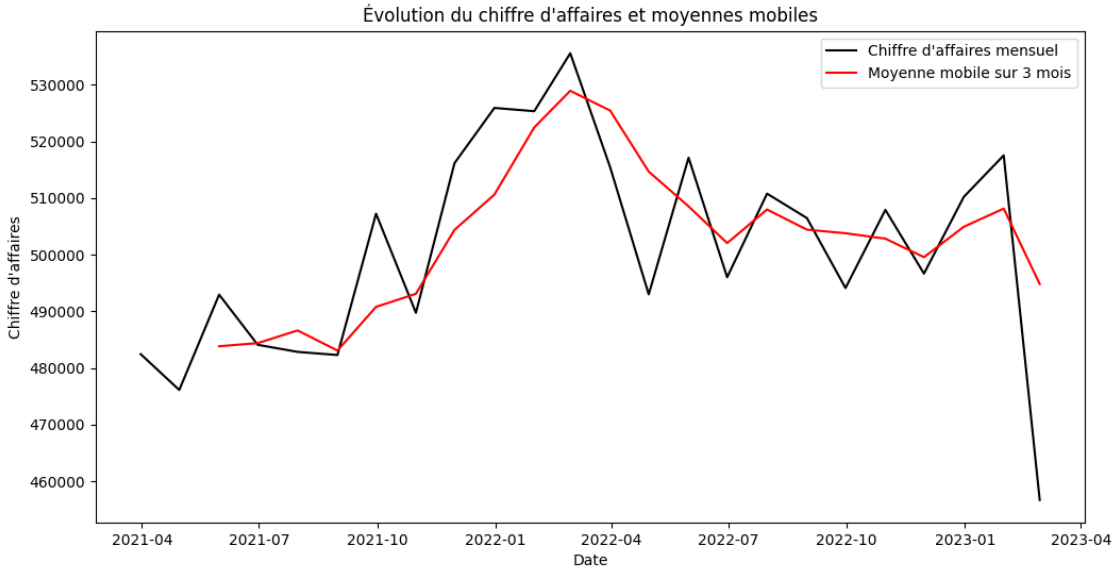
[72]:
```
monthly_sales = chiffre_affaires_par_mois.resample('M', on='date')['price'].
 ↪sum()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [2, 3, 5]
colors = ['red', 'blue', 'green']  # Couleurs pour les différentes moyennes␣
 ↪mobiles

# Afficher le graphique
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales, label='Chiffre d\'affaires mensuel', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    monthly_sales_rolling = monthly_sales.rolling(window=window).mean()
    plt.plot(monthly_sales_rolling, color=color, label=f'Moyenne mobile sur␣
 ↪{window} mois')
```

```
plt.title('Évolution du chiffre d\'affaires et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Chiffre d\'affaires')
plt.legend()
plt.show()
```



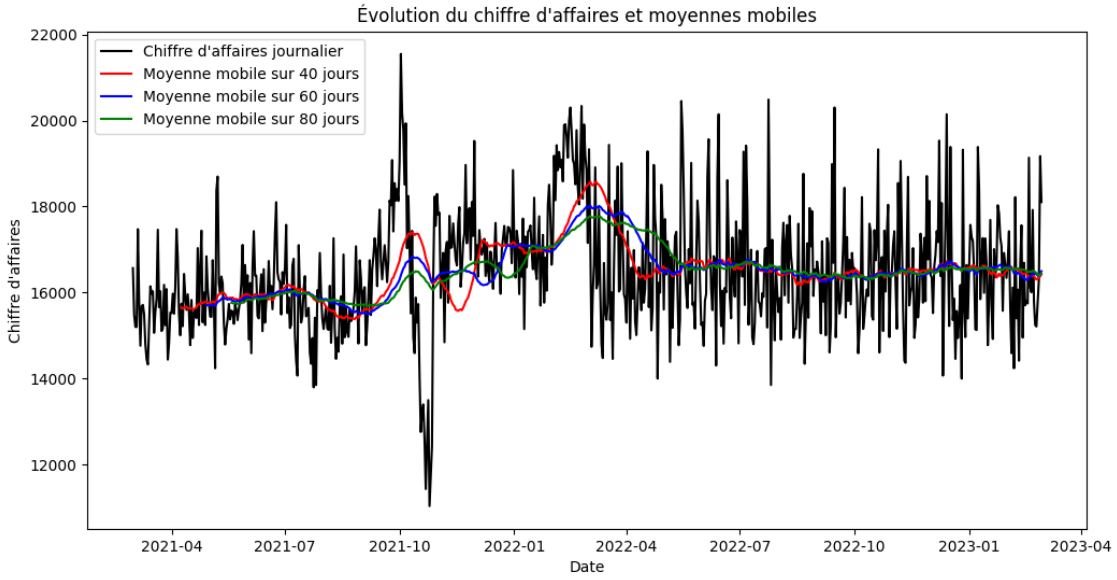
[73]:
```
monthly_sales = chiffre_affaires_par_mois.resample('M', on='date')['price'].
 ↪sum()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [3]
colors = ['red']

# Afficher le graphique
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales, label='Chiffre d\'affaires mensuel', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    monthly_sales_rolling = monthly_sales.rolling(window=window).mean()
    plt.plot(monthly_sales_rolling, color=color, label=f'Moyenne mobile sur␣
 ↪{window} mois')

plt.title('Évolution du chiffre d\'affaires et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Chiffre d\'affaires')
plt.legend()
```

```
plt.savefig('Moyenne_mobile_mois_chiffre_affaire.png', transparent=True)
plt.show()
```

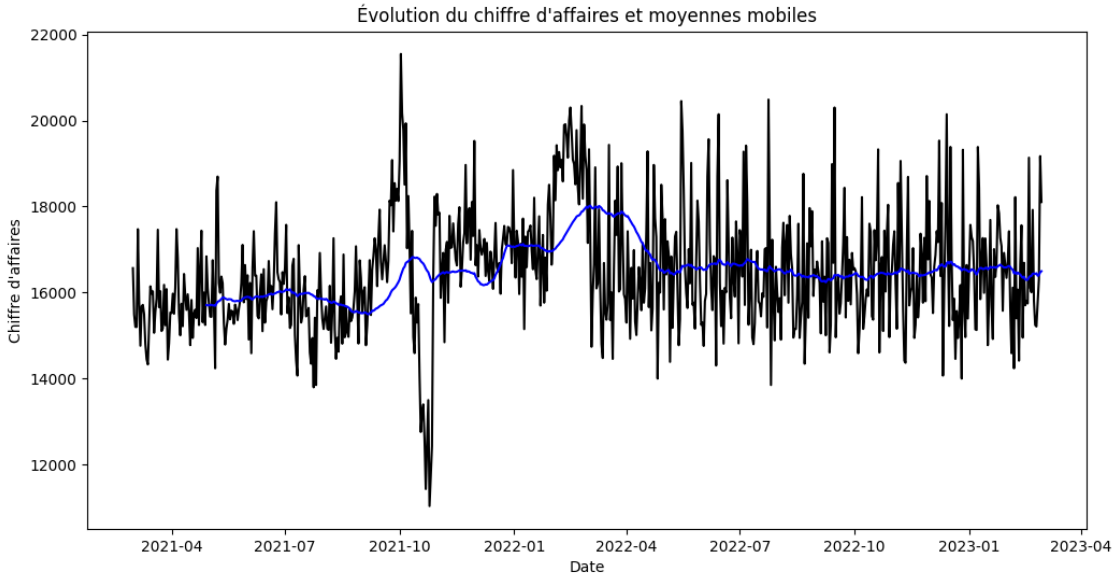


Par jour

[74]:
```
daily_sales = chiffre_affaires_par_jour.resample('D', on='date')['price'].sum()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [40, 60, 80]
colors = ['red', 'blue', 'green']

# Afficher le graphique
plt.figure(figsize=(12, 6))
plt.plot(daily_sales, label='Chiffre d\'affaires journalier', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    daily_sales_rolling = daily_sales.rolling(window=window).mean()
    plt.plot(daily_sales_rolling, color=color, label=f'Moyenne mobile sur␣
 ↪{window} jours')

plt.title('Évolution du chiffre d\'affaires et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Chiffre d\'affaires')
plt.legend()
plt.show()
```



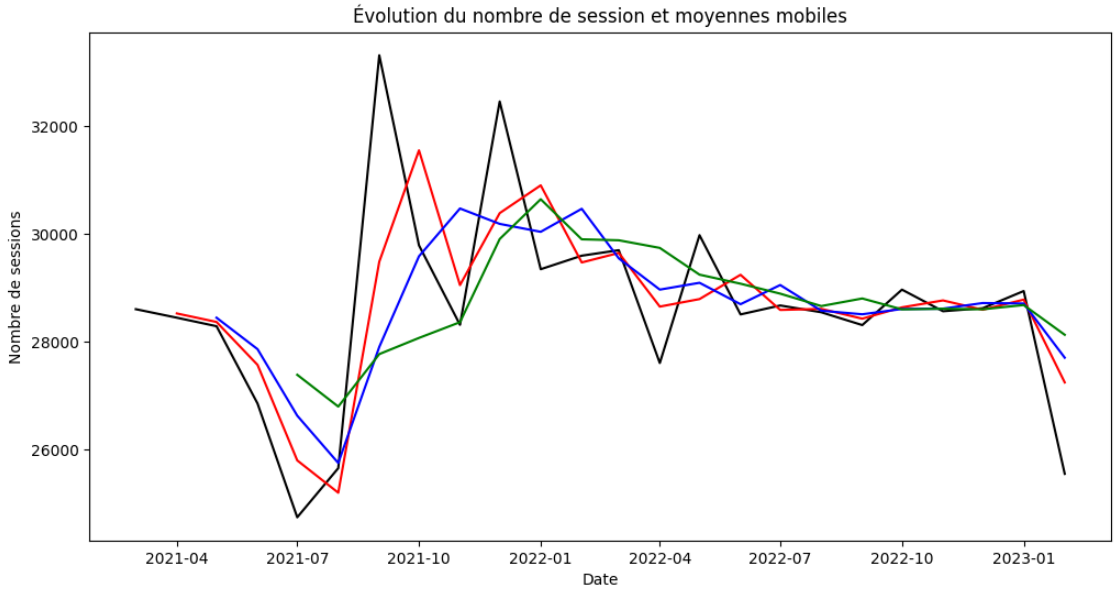
[75]:
```
daily_sales = chiffre_affaires_par_jour.resample('D', on='date')['price'].sum()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [60]
colors = ['blue']
# Afficher le graphique
plt.figure(figsize=(12, 6))
plt.plot(daily_sales, label='Chiffre d\'affaires journalier', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    daily_sales_rolling = daily_sales.rolling(window=window).mean()
    plt.plot(daily_sales_rolling, color=color, label=f'Moyenne mobile sur␣
 ↪{window} jours')

plt.title('Évolution du chiffre d\'affaires et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Chiffre d\'affaires')
plt.savefig('Moyenne_mobile_jour_chiffre_affaire.png', transparent=True)
plt.show()
```



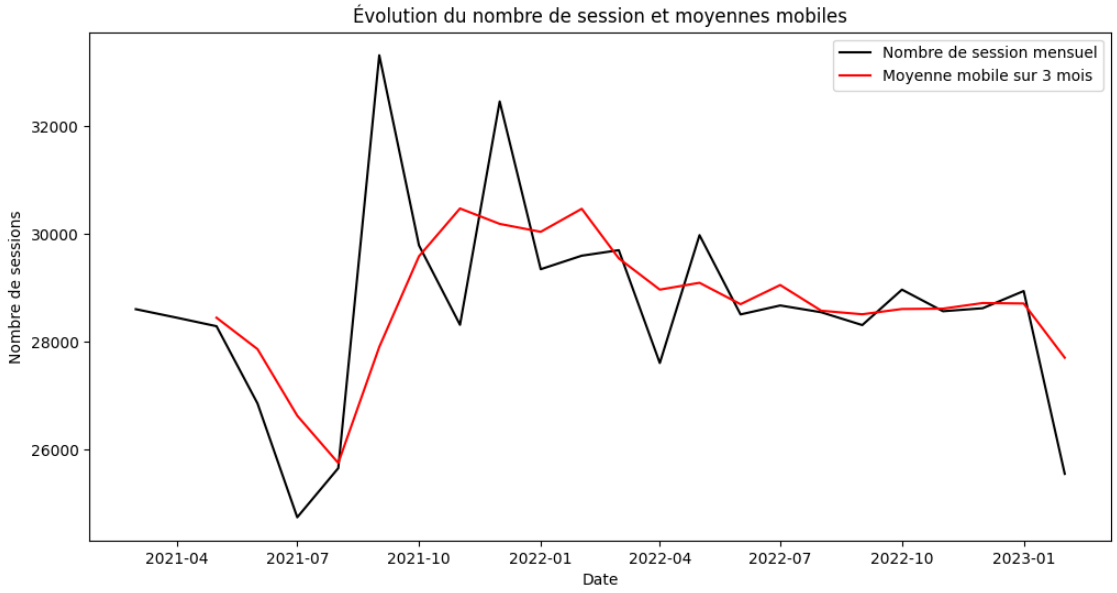
7.1.2 - Nombre sessions

Par mois

[76]:
```
sessions_par_mois = lapage.groupby(lapage['date'].dt.to_period("M")).
 ↪agg({'session_id': 'count'}).reset_index()

sessions_par_mois['date'] = sessions_par_mois['date'].dt.to_timestamp()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [2, 3, 5]
colors = ['red', 'blue', 'green']
plt.figure(figsize=(12, 6))
plt.plot(sessions_par_mois['date'], sessions_par_mois['session_id'],␣
 ↪label='Nombre de session mensuel', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    monthly_session_rolling = sessions_par_mois['session_id'].
 ↪rolling(window=window).mean()
    plt.plot(sessions_par_mois['date'], monthly_session_rolling, color=color,␣
 ↪label=f'Moyenne mobile sur {window} mois')

plt.title('Évolution du nombre de session et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Nombre de sessions')
plt.show()
```

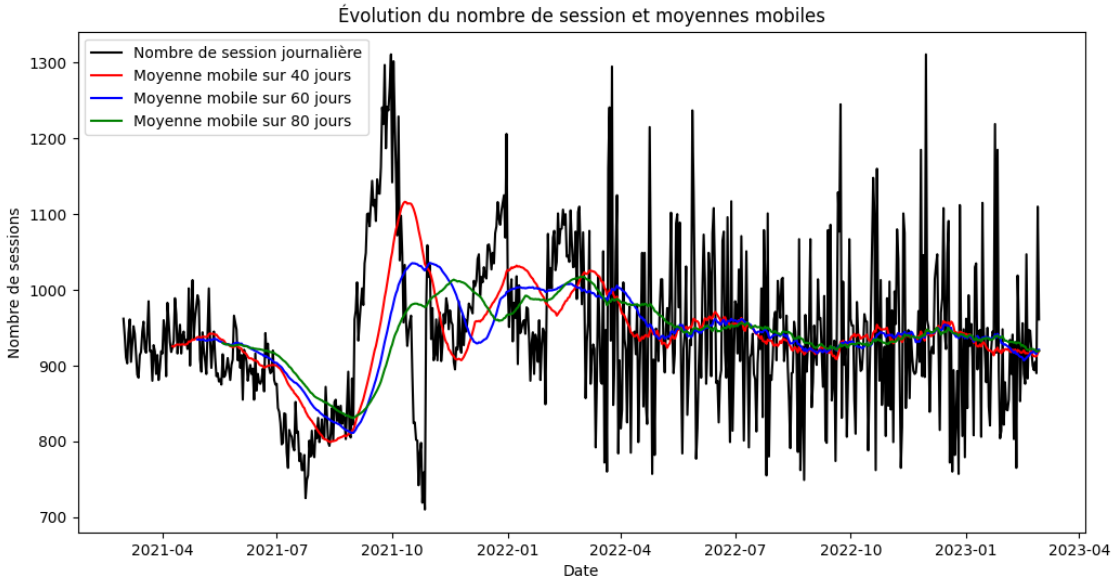


[77]:
```
sessions_par_mois = lapage.groupby(lapage['date'].dt.to_period("M")).
 ↪agg({'session_id': 'count'}).reset_index()

sessions_par_mois['date'] = sessions_par_mois['date'].dt.to_timestamp()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [3]
colors = ['red']
plt.figure(figsize=(12, 6))
plt.plot(sessions_par_mois['date'], sessions_par_mois['session_id'],␣
 ↪label='Nombre de session mensuel', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    monthly_session_rolling = sessions_par_mois['session_id'].
 ↪rolling(window=window).mean()
    plt.plot(sessions_par_mois['date'], monthly_session_rolling, color=color,␣
 ↪label=f'Moyenne mobile sur {window} mois')

plt.title('Évolution du nombre de session et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Nombre de sessions')
plt.legend()
plt.savefig('Moyenne_mobile_mois_session.png', transparent=True)
plt.show()
```

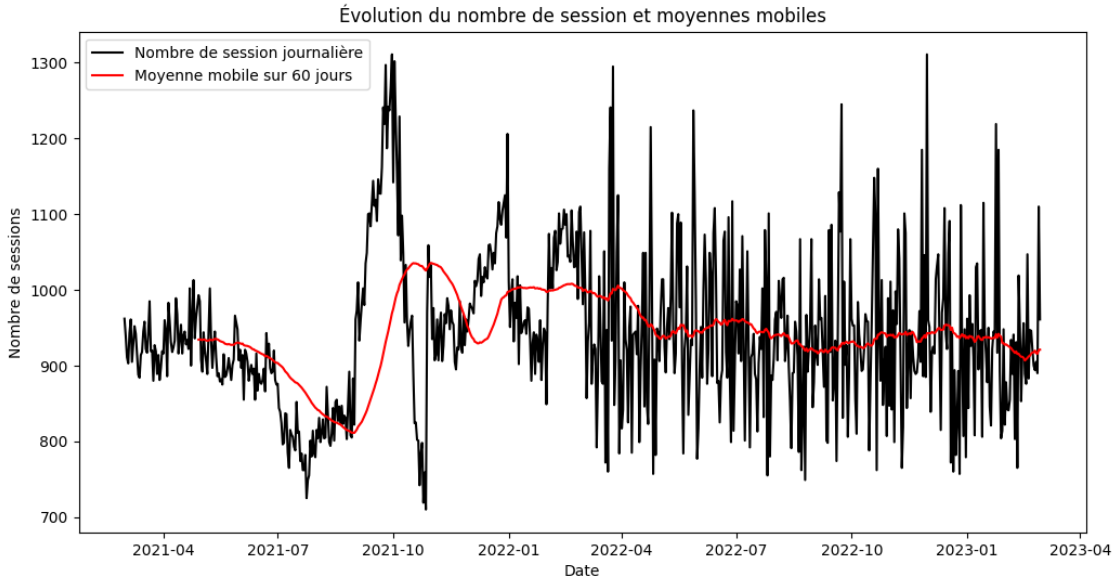


Par jour

[78]:
```
sessions_par_jour = lapage.groupby(lapage['date'].dt.to_period("D")).
 ↪agg({'session_id': 'count'}).reset_index()

sessions_par_jour['date'] = sessions_par_jour['date'].dt.to_timestamp()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [40, 60, 80]
colors = ['red', 'blue', 'green']
plt.figure(figsize=(12, 6))
plt.plot(sessions_par_jour['date'], sessions_par_jour['session_id'],␣
 ↪label='Nombre de session journalière', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    daily_session_rolling = sessions_par_jour['session_id'].
 ↪rolling(window=window).mean()
    plt.plot(sessions_par_jour['date'], daily_session_rolling, color=color,␣
 ↪label=f'Moyenne mobile sur {window} jours')

plt.title('Évolution du nombre de session et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Nombre de sessions')
plt.legend()
plt.show()
```



[79]:
```
sessions_par_jour = lapage.groupby(lapage['date'].dt.to_period("D")).
 ↪agg({'session_id': 'count'}).reset_index()

sessions_par_jour['date'] = sessions_par_jour['date'].dt.to_timestamp()

# Calculer les moyennes mobiles pour différentes fenêtres
rolling_windows = [60]
colors = ['red']
plt.figure(figsize=(12, 6))
plt.plot(sessions_par_jour['date'], sessions_par_jour['session_id'],␣
 ↪label='Nombre de session journalière', color='black')

# Tracer chaque moyenne mobile avec une couleur différente
for window, color in zip(rolling_windows, colors):
    daily_session_rolling = sessions_par_jour['session_id'].
 ↪rolling(window=window).mean()
    plt.plot(sessions_par_jour['date'], daily_session_rolling, color=color,␣
 ↪label=f'Moyenne mobile sur {window} jours')

plt.title('Évolution du nombre de session et moyennes mobiles')
plt.xlabel('Date')
plt.ylabel('Nombre de sessions')
plt.legend()
plt.savefig('Moyenne_mobile_jour_session.png', transparent=True)
plt.show()
```



7.2 - Analyse des catégories de produit

7.2.1 - Top des produits

[80]:
```
# Calcul du chiffre d'affaires par produit
sales_per_product = lapage.groupby('id_prod')['price'].sum().
 ↪sort_values(ascending=False)

# Afficher les tops produits
top_products = sales_per_product.head(10)
print("Top 10 des produits (par chiffre d'affaires):\n", top_products)
```

```
Top 10 des produits (par chiffre d'affaires):
 id_prod
2_159    94893.50
2_135    69334.95
2_112    65407.76
2_102    60736.78
2_209    56971.86
1_395    56617.47
1_369    56136.60
2_110    53846.25
1_383    53834.43
1_414    53522.18
Name: price, dtype: float64
```

7.2.2 - Flop des produits

[81]:
```
# Afficher les flops produits
flop_products = sales_per_product.tail(10)
print("Flop 10 des produits (par chiffre d'affaires):\n", flop_products)
```

```
Flop 10 des produits (par chiffre d'affaires):
 id_prod
0_1840    2.56
0_898     2.54
0_1498    2.48
0_1728    2.27
0_807     1.99
0_1601    1.99
0_541     1.99
0_1653    1.98
0_1284    1.38
0_1539    0.99
Name: price, dtype: float64
```

6.5.3 - Top et Flop des produits par catégorie

[82]:
```
for category in lapage['categ'].unique():
    print(f"\nCatégorie {category} :")
    category_data = lapage[lapage['categ'] == category]

    # Top produits dans la catégorie
    top_products_category = category_data.groupby('id_prod')['price'].sum().
 ↪sort_values(ascending=False).head(5)
    print("Top 5 des produits:\n", top_products_category)

    # Flop produits dans la catégorie
    flop_products_category = category_data.groupby('id_prod')['price'].sum().
 ↪sort_values(ascending=True).head(5)
    print("Flop 5 des produits:\n", flop_products_category)
```

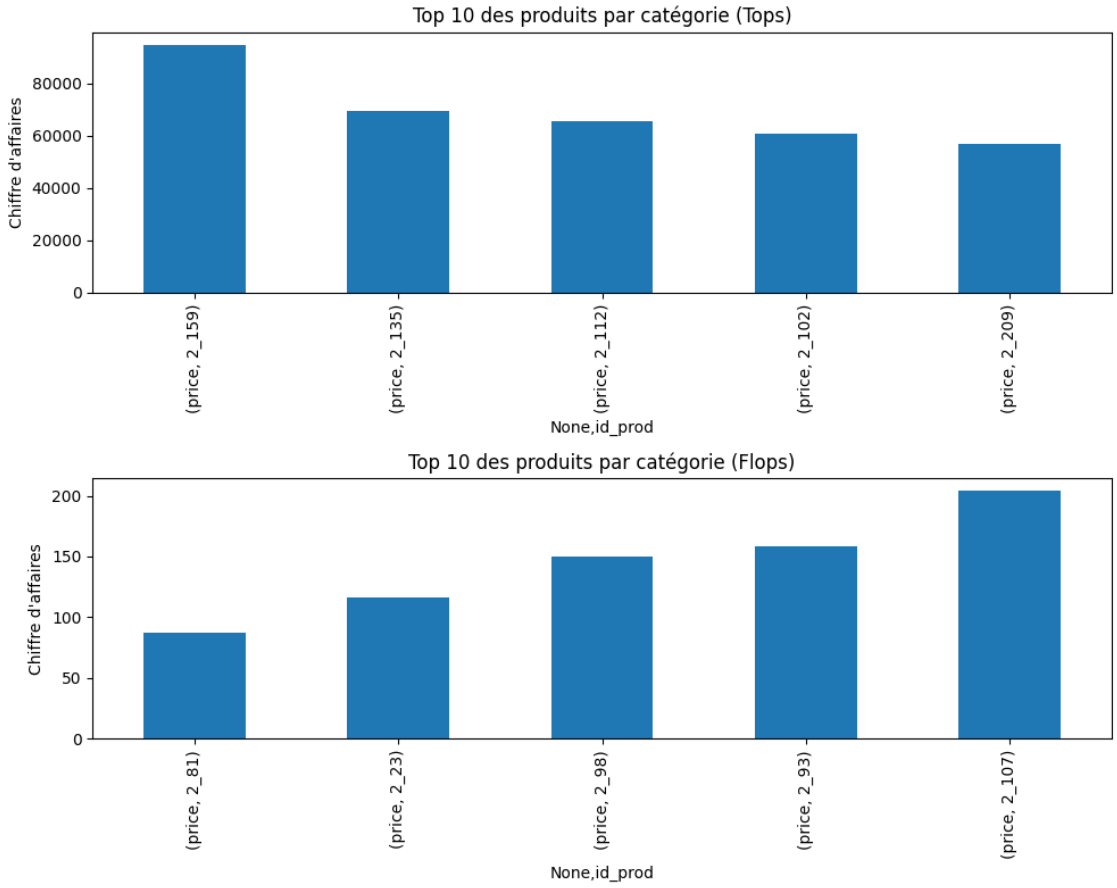
```
Catégorie 0.0 :
Top 5 des produits:
 id_prod
0_1441    23452.65
0_1421    23008.49
0_1414    22287.00
0_1451    21689.15
0_1417    21372.12
Name: price, dtype: float64
Flop 5 des produits:
 id_prod
0_1539    0.99
0_1284    1.38
0_1653    1.98
```

```
0_807     1.99
0_1601    1.99
Name: price, dtype: float64

Catégorie 1.0 :
Top 5 des produits:
 id_prod
1_395    56617.47
1_369    56136.60
1_383    53834.43
1_414    53522.18
1_498    51460.74
Name: price, dtype: float64
Flop 5 des produits:
 id_prod
1_420    14.24
1_224    19.80
1_470    21.64
1_473    26.91
1_404    29.55
Name: price, dtype: float64

Catégorie 2.0 :
Top 5 des produits:
 id_prod
2_159    94893.50
2_135    69334.95
2_112    65407.76
2_102    60736.78
2_209    56971.86
Name: price, dtype: float64
Flop 5 des produits:
 id_prod
2_81      86.99
2_23     115.99
2_98     149.74
2_93     157.98
2_107    203.94
Name: price, dtype: float64
```

7.2.3 - la répartition par catégorie

[83]:
```
# Visualisation des tops et flops par catégorie
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))

top_products_category.reset_index().set_index('id_prod').unstack().
 ↪plot(kind='bar', ax=axes[0], title="Top 10 des produits par catégorie␣
 ↪(Tops)")
```

```
axes[0].set_ylabel("Chiffre d'affaires")

flop_products_category.reset_index().set_index('id_prod').unstack().
 ↪plot(kind='bar', ax=axes[1], title="Top 10 des produits par catégorie␣
 ↪(Flops)")
axes[1].set_ylabel("Chiffre d'affaires")

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



<h2 style="font-weight: bold; color: #001F3F;">Etape 8 - analyse des profils client</h2>

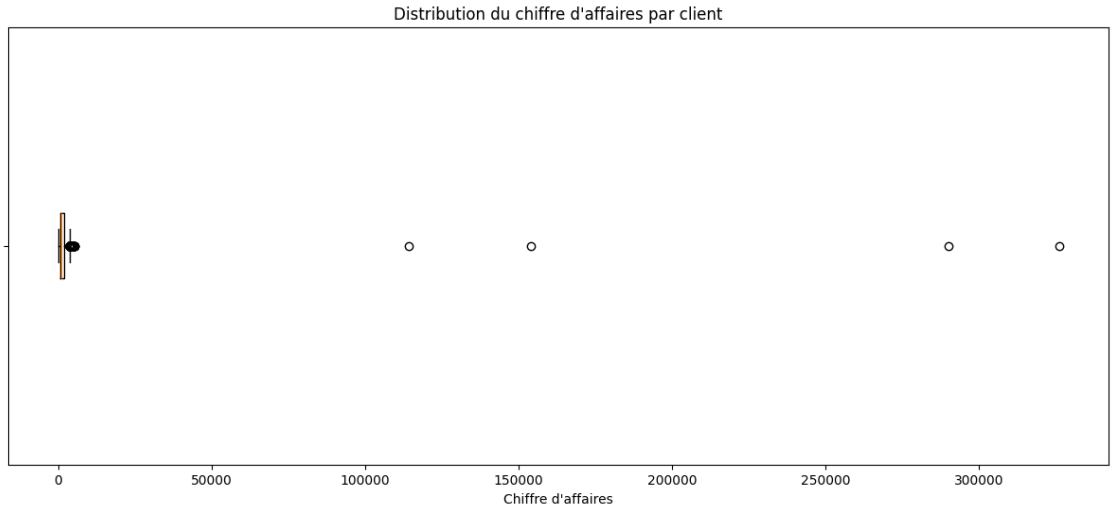
8.1 - courbe de Lorenz chiffre d'affaire par client

[84]:
```
# Function to calculate the Gini coefficient
def gini(x):
    n = len(x)
    S = x.sum()
    r = np.argsort(np.argsort(-x))
    return 1 - (2.0 * (r * x).sum() + S) / (n * S)
```

```
# Calculating the Lorenz curve
dataset_lorenz = lapage.groupby('client_id').agg({'price':'sum'}).reset_index()
prix = dataset_lorenz['price'].values
prix_sorted = np.sort(prix)
lorenz_prix = np.cumsum(prix_sorted) / prix_sorted.sum()
x = np.linspace(0, 1, len(lorenz_prix))
y = lorenz_prix

# Calculating the Gini coefficient
indice_gini = gini(prix)

# Plotting the Lorenz curve
fig = px.area(x=x, y=y)
fig.add_scatter(x=[0, 1], y=[0, 1], mode='lines', line=dict(color='red'),␣
 ↪name='Distribution égalitaire')
fig.add_shape(
        dict(
            type="line",
            x0=.5,
            y0=0,
            x1=.5,
            y1=.5,
            line=dict(color="blue", dash="dash"),
            xref='x',
            yref='y',
            name = "autre ligne"
        ))

fig.update_layout(
        xaxis_title="Distribution des clients (%)",
        yaxis_title="Somme cumulée du chiffre d'affaires (%)",
        title=f"Courbe de Lorenz du chiffre d'affaires par client <br><sup>␣
 ↪Indice de GINI : {round(indice_gini,2)}</sup>",
    )
```

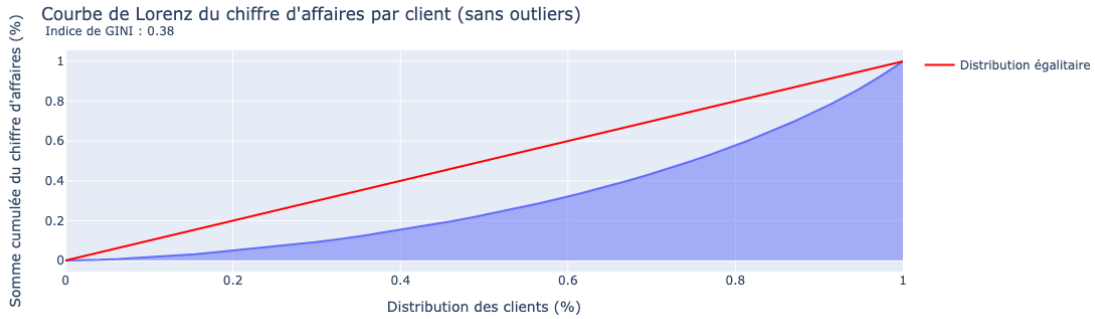
8.1.1 - Courbe de lorenz excluant les outliers

[85]:
```
# Calcul du chiffre d'affaires par client
revenue_per_customer = lapage.groupby('client_id')['price'].sum()

# Affichage du boxplot pour identifier les outliers
plt.figure(figsize=(15, 6))
plt.boxplot(revenue_per_customer.values, vert=False)
plt.yticks([1], [""], rotation=45)
plt.title('Distribution du chiffre d\'affaires par client')
plt.xlabel('Chiffre d\'affaires')
plt.show()

# Identification des outliers
Q1 = np.percentile(revenue_per_customer, 25)
Q3 = np.percentile(revenue_per_customer, 75)
IQR = Q3 - Q1
outlier_threshold = Q3 + 1.5 * IQR
```



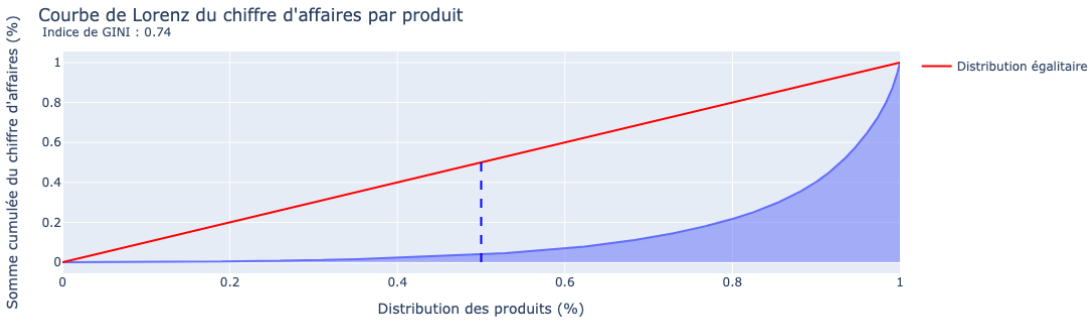
[86]:
```
def gini(x):
    n = len(x)
    S = x.sum()
    r = np.argsort(np.argsort(-x))
    return 1 - (2.0 * (r * x).sum() + S) / (n * S)

# Calcul du chiffre d'affaires par client
dataset_lorenz = lapage.groupby('client_id').agg({'price':'sum'}).reset_index()
prix = dataset_lorenz['price'].values
```

```
# Identification et exclusion des outliers
Q1 = np.percentile(prix, 25)
Q3 = np.percentile(prix, 75)
IQR = Q3 - Q1
outlier_threshold = Q3 + 1.5 * IQR
prix_filtered = np.sort(prix[prix <= outlier_threshold])

# Recalcul de la courbe de Lorenz avec les données filtrées
lorenz_prix_filtered = np.cumsum(prix_filtered) / prix_filtered.sum()
x_filtered = np.linspace(0, 1, len(lorenz_prix_filtered))
y_filtered = lorenz_prix_filtered

# Recalcul de l'indice de Gini avec les données filtrées
indice_gini_filtered = gini(prix_filtered)

# Tracé de la courbe de Lorenz filtrée
fig_filtered = px.area(x=x_filtered, y=y_filtered)
fig_filtered.add_scatter(x=[0, 1], y=[0, 1], mode='lines',␣
 ↪line=dict(color='red'), name='Distribution égalitaire')
fig_filtered.update_layout(
    xaxis_title="Distribution des clients (%)",
    yaxis_title="Somme cumulée du chiffre d'affaires (%)",
    title=f"Courbe de Lorenz du chiffre d'affaires par client (sans outliers)␣
 ↪<br><sup> Indice de GINI : {round(indice_gini_filtered, 2)}</sup>",
)

fig_filtered.show()
fig_filtered.write_image('courbelorenz_chiffre_affaire_client.png',␣
 ↪format='png', width=1600, height=900, scale=2)
```



8.2 - courbe de Lorenz chiffre d'affaire par produit

[87]:
```
# Function to calculate the Gini coefficient
def gini(x):
    n = len(x)
    S = x.sum()
    r = np.argsort(np.argsort(-x))
    return 1 - (2.0 * (r * x).sum() + S) / (n * S)

# Calculating the Lorenz curve
dataset_lorenz = lapage.groupby('id_prod').agg({'price':'sum'}).reset_index()
prix = dataset_lorenz['price'].values
prix_sorted = np.sort(prix)
lorenz_prix = np.cumsum(prix_sorted) / prix_sorted.sum()
x = np.linspace(0, 1, len(lorenz_prix))
y = lorenz_prix

# Calculating the Gini coefficient
indice_gini = gini(prix)

# Plotting the Lorenz curve
fig = px.area(x=x, y=y)
fig.add_scatter(x=[0, 1], y=[0, 1], mode='lines', line=dict(color='red'),␣
 ↪name='Distribution égalitaire')
fig.add_shape(
        dict(
            type="line",
            x0=.5,
            y0=0,
            x1=.5,
            y1=.5,
            line=dict(color="blue", dash="dash"),
            xref='x',
            yref='y',
            name = "autre ligne"
        ))

fig.update_layout(
        xaxis_title="Distribution des produits (%)",
        yaxis_title="Somme cumulée du chiffre d'affaires (%)",
        title=f"Courbe de Lorenz du chiffre d'affaires par produit <br><sup>␣
 ↪Indice de GINI : {round(indice_gini,2)}</sup>",
    )

fig.show()
fig.write_image('courbelorenz_chiffre_affaire.png', format='png', width=1600,␣
 ↪height=900, scale=2)
```



<h2 style="font-weight: bold; color: #001F3F;">Etape 9 - Etude du comportement des clients</h2>

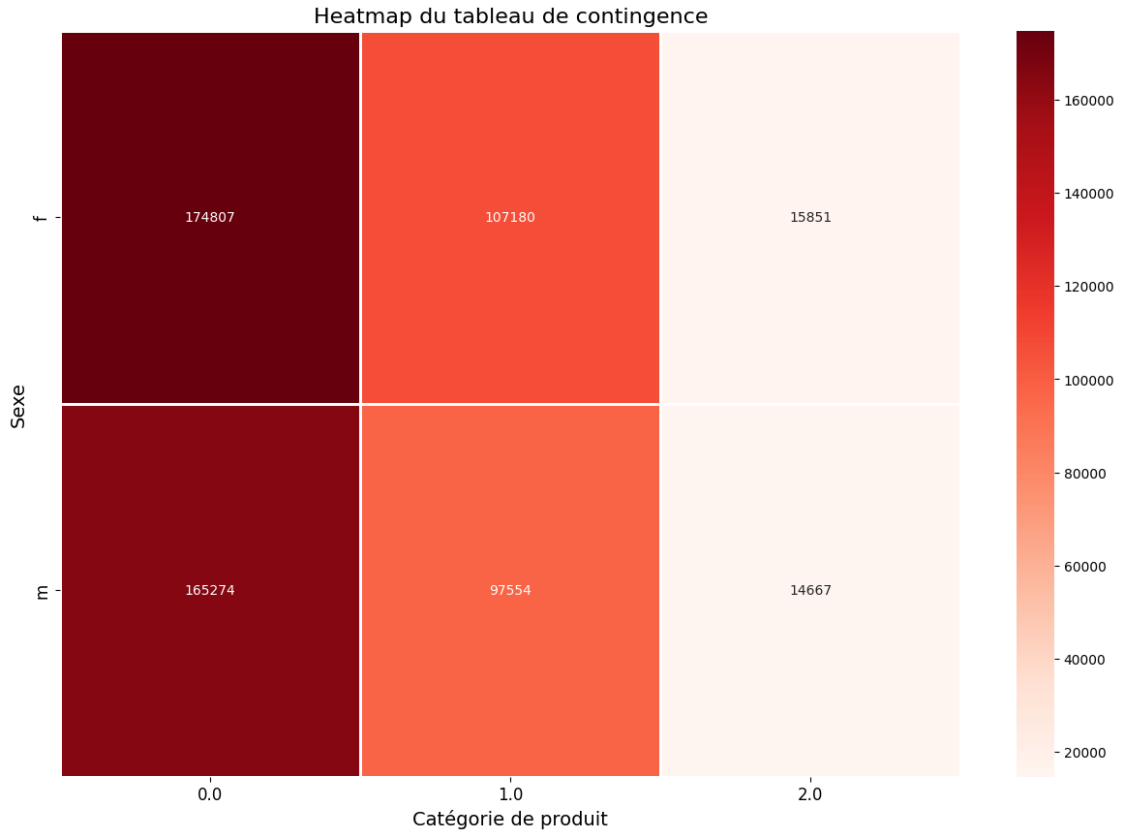
9.1 - Lien entre le genre du client et les catégories des livres achetés

9.1.1 - Test de chi carré

[88]:
```
# Calcul du chiffre d'affaires par client
dataset_lorenz = lapage.groupby('client_id').agg({'price':'sum'}).reset_index()
prix = dataset_lorenz['price'].values

# Identification et exclusion des outliers
Q1 = np.percentile(prix, 25)
Q3 = np.percentile(prix, 75)
IQR = Q3 - Q1
outlier_threshold = Q3 + 1.5 * IQR
prix_filtered = np.sort(prix[prix <= outlier_threshold])

# Filtrer le DataFrame original pour exclure les clients avec des valeurs␣
 ↪d'achat d'outliers
lapage = lapage[lapage['client_id'].isin(dataset_lorenz[prix <=␣
 ↪outlier_threshold]['client_id'])]
```

[89]:
```
from scipy.stats import chi2_contingency

# Création de la table de contingence
crosstab = pd.crosstab(lapage['sex'], lapage['categ'])

# Configuration de la taille de la figure
plt.subplots(figsize=(15, 10))
plt.tick_params(axis='both', which='major', labelsize=12, labelbottom=True,␣
 ↪bottom=True, top=False, labeltop=False)

# Création du heatmap
sns.heatmap(crosstab, cmap="Reds", linewidth=1, annot=True, fmt="d",␣
 ↪xticklabels=True, yticklabels=True)
```

```
plt.title('Heatmap du tableau de contingence', fontsize=16)
plt.xlabel('Catégorie de produit', fontsize=14)
plt.ylabel('Sexe', fontsize=14)

plt.savefig('heatmap.png', transparent=True, bbox_inches='tight')
plt.show()

# Effectuer le test du Chi-carré
chi2, p, _, _ = chi2_contingency(crosstab)

# Afficher les résultats du test du Chi-carré
print("\nRésultats du test du Chi-carré :")
print("Chi2 Statistic:", chi2)
print("p-value:", p)

# Afficher l'interprétation des résultats
if p < 0.05:
    print("\nIl y a une corrélation statistiquement significative entre le␣
 ↪genre et la catégorie des produits.")
else:
    print("\nIl n'y a pas de corrélation statistiquement significative entre le␣
 ↪genre et la catégorie des produits.")
```



```
Résultats du test du Chi-carré :
Chi2 Statistic: 46.50393049579011
p-value: 7.976269133762762e-11

Il y a une corrélation statistiquement significative entre le genre et la
catégorie des produits.
```

9.2 - Lien entre l'âge du client et le montant total des achats

9.2.1 - Calcul de l'âge des clients

[90]:
```
maintenant = datetime.now()
annee = maintenant.year

lapage['age'] = annee - lapage['birth']

# Afficher les premières lignes du DataFrame pour vérifier
lapage.head()
```

[90]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age
0	0_1259	2021-03-01 00:01:07.843138	s_1	c_329	11.99	0.0	f	1967	57
1	0_1259	2022-10-01 00:01:07.843138	s_275943	c_329	11.99	0.0	f	1967	57
2	0_1259	2022-12-01 00:01:07.843138	s_305291	c_329	11.99	0.0	f	1967	57
3	0_1259	2023-01-01 00:01:07.843138	s_320153	c_329	11.99	0.0	f	1967	57
4	1_397	2021-11-23 18:21:56.361813	s_123998	c_329	18.99	1.0	f	1967	57

9.2.2 - Graphique nuage point

[91]:
```
# Calculer Q1, Q3 et IQR pour les montants totaux des achats par âge
Q1_age = total_achats_par_age['price'].quantile(0.25)
Q3_age = total_achats_par_age['price'].quantile(0.75)
IQR_age = Q3_age - Q1_age

# Définir le seuil des valeurs aberrantes
seuil_outlier_age = Q3_age + 1.5 * IQR_age

# Filtrer les valeurs aberrantes basées sur l'âge
total_achats_par_age_sans_outliers =␣
 ↪total_achats_par_age[total_achats_par_age['price'] <= seuil_outlier_age]
```

```
# Générer le graphique sans les valeurs aberrantes
plt.figure(figsize=(10, 6))
sns.scatterplot(data=total_achats_par_age_sans_outliers, x='age', y='price')
plt.title('Lien entre l\'âge du client et le montant total des achats (sans␣
 ↪outliers)')
plt.xlabel('Âge du Client')
plt.ylabel('Montant Total des Achats')
plt.savefig("nuage_de_points_montant_total.png")
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[91], line 2
      1 # Calculer Q1, Q3 et IQR pour les montants totaux des achats par âge
----> 2 Q1_age = total_achats_par_age['price'].quantile(0.25)
      3 Q3_age = total_achats_par_age['price'].quantile(0.75)
      4 IQR_age = Q3_age - Q1_age

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

9.2.3 - Corrélation entre les deux variables

[92]:
```
# Calcul de la corrélation de Spearman entre l'âge et le montant total des␣
 ↪achats
correlation, p_value = statspearmanr(total_achats_par_age_sans_outliers['age'],␣
 ↪total_achats_par_age_sans_outliers['price'])

# Affichage du coefficient de corrélation de Spearman et de la valeur p
print(f"Coefficient de corrélation de Spearman : {correlation:.2f}")
print(f"Valeur p : {p_value:.3f}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[92], line 2
      1 # Calcul de la corrélation de Spearman entre l'âge et le montant total␣
 ↪des achats
----> 2 correlation, p_value =␣
 ↪spearmanr(total_achats_par_age_sans_outliers['age'],␣
 ↪total_achats_par_age_sans_outliers['price'])
      4 # Affichage du coefficient de corrélation de Spearman et de la valeur p
      5 print(f"Coefficient de corrélation de Spearman : {correlation:.2f}")

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

9.3 - Lien entre l'âge du client et la fréquence d'achat

9.3.1 - Calcul de la fréquence d'achat

[ ]:
```
# Calculer la fréquence d'achat par âge
age_frequency = lapage.groupby('age').size().reset_index(name='frequency')

# Créer le graphique en nuage de points
plt.figure(figsize=(10, 6))
plt.scatter(age_frequency['age'], age_frequency['frequency'], color='blue',␣
 ↪alpha=0.5)
plt.title('Lien entre l\'âge du client et la fréquence d\'achat')
plt.xlabel('Âge du Client')
plt.ylabel('Fréquence d\'Achat')
plt.grid(True)
plt.savefig("nuage_de_points_age_achats_frequence_achat.png")
plt.show()
```

9.3.2 - Corrélation entre les deux variables

[ ]:
```
correlation, p_value = spearmanr(age_frequency['age'],␣
 ↪age_frequency['frequency'])

# Afficher le coefficient de corrélation de Spearman et la valeur p
print(f"Coefficient de corrélation de Spearman : {correlation:.2f}")
print(f"Valeur p associée : {p_value:.3f}")
```

9.4 - Lien entre l'âge du client et la taille du panier moyen

9.4.1 - Calcul du panier moyen

[ ]:
```
panier_moyen = lapage.groupby(['age', 'session_id']).count().reset_index()

# Seconde agrégation selon l'âge du client en moyenne de produits achetés
panier_moyen = panier_moyen.groupby('age')['id_prod'].mean().reset_index()

# Visualisation avec un scatterplot (âge client vs taille panier moyen)
plt.scatter(panier_moyen['age'], panier_moyen['id_prod'], color='purple')
plt.xlabel('Âge')
plt.ylabel('Panier moyen (Nombre de produits)')
plt.title('Panier moyen en nombre de produits selon l\'âge du client')
plt.show()
```

9.4.2 - Corrélation entre les deux variables

[ ]:
```
correlation, p_value = pearsonr(panier_moyen['age'], panier_moyen['id_prod'])
print(f"Corrélation de Pearson : {correlation:.2f}")
print(f"P-value : {p_value:.3f}")
```

9.5 - Lien entre l'âge du client et la catégorie des livres achetés

9.5.1 - Graphique Boxplot

[ ]:
```
# Créer un boxplot
plt.figure(figsize=(12, 6))
plt.boxplot([lapage[lapage['categ'] == 0]['age'], lapage[lapage['categ'] ==␣
 ↪1]['age'], lapage[lapage['categ'] == 2]['age']], labels=['Catégorie 0',␣
 ↪'Catégorie 1', 'Catégorie 2'])
plt.title("Lien entre l'âge du client et la catégorie des livres achetés")
plt.xlabel("Catégorie de Livre")
plt.ylabel("Âge du Client")

plt.savefig("boxplot_age_categorie.png")
plt.show()
```

9.5.2 - Corrélation

[ ]:
```
age_categorie_0 = lapage[lapage['categ'] == 0]['age']
age_categorie_1 = lapage[lapage['categ'] == 1]['age']
age_categorie_2 = lapage[lapage['categ'] == 2]['age']

# Effectuez le test ANOVA
f_statistic, p_value = f_oneway(age_categorie_0, age_categorie_1,␣
 ↪age_categorie_2)

print("Résultats du test ANOVA :")
print("Statistique F :", f_statistic)
print("P-valeur :", p_value)

alpha = 0.05
if p_value < alpha:
    print("\nIl y a des différences significatives entre les âges des clients␣
 ↪dans les catégories de livres.")
else:
    print("\nIl n'y a pas de différences significatives entre les âges des␣
 ↪clients dans les catégories de livres.")
```

[ ]:
```

```