

✎ Analyse de l'entreprise "Lapage"

✎ I. Téléchargement données et bibliothèques

✎ A. Chargement bibliothèques

```
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
```


✎ B. Téléchargement des données

```
# Monter le drive
from google.colab import drive
drive.mount('/content/drive')
chemin="//content//drive//My Drive//Projet 9"
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_re

```
path="//content//drive//My Drive//Projet 9"
# Téléchargement du fichier customers.csv
df_customer=pd.read_csv(path+"//customers.csv",sep=";")
#Téléchargement de products.csv
df_product=pd.read_csv(path+"//products.csv",sep=";")
"""
```



```
#telechargement de transactions.csv
df_transactions=pd.read_csv(path+"//Transactions.csv",sep=";")
```

 <ipython-input-5-2460340464>:7: DtypeWarning: Columns (0,1,2,3) have mixed types. Specify dtype option on import or
df_transactions=pd.read_csv(path+"//Transactions.csv",sep=";")

✓ II. Analyse exploratoire des fichiers

✓ A. Exploration de customers

```
df_customer.head()
```

| | 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 |

Étapes suivantes : [Générer du code avec df_customer](#) [Afficher les graphiques recommandés](#) [New interactive sheet](#)

```
print(f"customers contient {df_customer.shape[0]} lignes.")
```

 customers contient 8621 lignes.

```
print("Information sur les colonnes :")
df_customer.info()
```



```
Information sur les colonnes :
<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
```

Colonne client_id

```
print(f"Nombre de valeurs dupliqués : {df_customer['client_id'].duplicated().sum()}")
```

```
Nombre de valeurs dupliqués : 0
```

Colonne "sex"

```
#Lister les valeurs de la colonne sex
print("Liste des valeurs de la colonne sex : ")
print(df_customer["sex"].unique().tolist())
```

```
Liste des valeurs de la colonne sex :
['f', 'm']
```

Colonne "birth"

```
# Respect du format pour les années(XXXX)
nb_invalide_birth = df_customer[~df_customer["birth"].astype(str).str.match(r"^\d{4}$")].shape[0]

print(f"Nombre de valeurs invalides dans 'birth' : {nb_invalide_birth}")
```

Nombre de valeurs invalides dans 'birth' : 0

```
df_customer.head()
```

| | 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 |



Étapes suivantes :

[Générer du code avec df_customer](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

Création Colonne age

```
from dateutil.relativedelta import relativedelta

def convert_to_datetime(year_series):
    dates = pd.to_datetime(year_series.astype(str) + '-01-01', errors='coerce')

    if dates.dt.tz is not None:
        dates = dates.dt.tz_convert(None)
    return dates

df_customer['birth'] = convert_to_datetime(df_customer['birth'])
print("Vérification des dates converties:")
```

```

print(df_customer['birth'].head())

def calculate_age(birth_date):
    today = datetime.now()
    try:
        return today.year - birth_date.year - ((today.month, today.day) < (birth_date.month, birth_date.day))
    except:
        return None

df_customer['age'] = df_customer['birth'].apply(calculate_age)
# Filtrage des valeurs aberrantes
valeurs_aberrantes = df_customer[(df_customer['age'] < 7) | (df_customer['age'] > 120)]
print(f"\nNombre de valeurs aberrantes: {len(valeurs_aberrantes)}")
print(valeurs_aberrantes[['birth', 'age']] if not valeurs_aberrantes.empty else "Aucune valeur aberrante")

print("\nRésultat final:")
print(df_customer.head())

```

Vérification des dates converties:

0 1967-01-01

1 1975-01-01

2 1984-01-01

3 1962-01-01

4 1943-01-01

Name: birth, dtype: datetime64[ns]

Nombre de valeurs aberrantes: 0

Aucune valeur aberrante

Résultat final:

| | client_id | sex | birth | age |
|---|-----------|-----|------------|-----|
| 0 | c_4410 | f | 1967-01-01 | 58 |
| 1 | c_7839 | f | 1975-01-01 | 50 |
| 2 | c_1699 | f | 1984-01-01 | 41 |
| 3 | c_5961 | f | 1962-01-01 | 63 |
| 4 | c_5370 | m | 1943-01-01 | 82 |

T 000000 III 12775 01 01 02

Étapes suivantes :

[Générer du code avec df_customer](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

```
print("'customer' est prêt à être utilisé.")
```

```
'customer' est prêt à être utilisé.
```

▼ B. Exploration de product

```
df_product.head()
```

Étapes suivantes :

[Générer du code avec df_product](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)`df_product.info()`

```
<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
```

Colonne *categ*

```
valeur_unique_categ=df_product["categ"].unique().tolist()
print(f"Liste des valeurs que prend categ : {valeur_unique_categ}")
```

```
Liste des valeurs que prend categ : [0, 1, 2]
```

Colonne *price*`df_product["price"].describe()`

▼ C. Exploration de transactions

```
df_transactions.head()
```

```
df_transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
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: 32.0+ MB

```

```

print(df_transactions["date"].isna().sum())
print(df_transactions["id_prod"].isna().sum())
print(df_transactions["session_id"].isna().sum())
print(df_transactions["client_id"].isna().sum())
print(f"Il y a des lignes où il n'y a aucune valeur {df_transactions['client_id'].isna().sum()}")

```

```

361041
361041
361041
361041
Il y a des lignes où il n'y a aucune valeur 361041, je vais donc les effacer

```

```

df_transactions = df_transactions.dropna(how='all')
print(f"Vérification: {df_transactions['date'].isna().sum()} lignes contenant Nan")
print(f"Nombre de ligne df_transactions: {df_transactions.shape[0]}")
print("df_transactions est prêt à être utilisé")

```

```

Vérification: 0 lignes contenant Nan
Nombre de ligne df_transactions: 687534
df_transactions est prêt à être utilisé

```

```

df_transactions["date"] = pd.to_datetime(df_transactions["date"])
df_transactions = df_transactions.sort_values(by='date')

```

```

...

```

▼ III) Jonction des tables

▼ A. Jonction intérieur transactions-cutomers

```
df_merge=df_transactions.merge(df_customer,left_on="client_id",right_on="client_id",how="inner")
df_merge.head()
```

▼ B. Jonction intérieur merge-products

```
df_merge=df_merge.merge(df_product,how="inner",right_on="id_prod",left_on="id_prod")
df_merge.head()
```

✎ IV) Analyse des indicateurs de performances

✎ 0) Analyse du prix et nombre article

✎ Prix général et nombre article général

```
prix_moyen = df_merge['price'].mean()
prix_median = df_merge["price"].median()

print(f"Prix moyen articles : {round(prix_moyen, 2)} euros.")
print(f"Médiane du prix articles : {prix_median} euros.")

if prix_moyen < prix_median:
    print(f"Asymétrie distribution à gauche.")
else:
    print(f"Asymétrie distribution à droite.")
df_merge["price"].describe()
```

```
plt.figure(figsize=(10,6))
sn.boxplot(x=df_merge["price"],color="skyblue")
plt.title("Distribution des prix des articles")
plt.xlabel("Prix")
plt.grid(axis='x',linestyle="--")
plt.show()
```

▼ Nombre d'article distincts vendus par rapport au nombre présent aux magasins

```
# liste produits vendus
produits_vendus_ids = df_merge["id_prod"].unique()

produits_distincts_vendus = len(produits_vendus_ids)
produits_distincts_presents_librairie = df_product.shape[0]

# produits sans ventes
produits_sans_ventes = df_product[~df_product["id_prod"].isin(produits_vendus_ids)]
nb_produits_sans_ventes = len(produits_sans_ventes)

print(f"Nombre de produits distincts vendus : {produits_distincts_vendus}/{produits_distincts_presents_librairie}")
print(f"Pourcentage sans ventes : {round(100-(produits_distincts_vendus/produits_distincts_presents_librairie*100), 2)}%")
print(f"Total articles non vendus : {nb_produits_sans_ventes}")

# Analyse par catégorie
if not produits_sans_ventes.empty:

    stats_sans_ventes = produits_sans_ventes.groupby("categ").size().reset_index(name='nb_sans_ventes')
    stats_total = df_product.groupby("categ").size().reset_index(name='nb_total')
```

```
stats_completes = stats_sans_ventes.merge(stats_total, on="categ")
stats_completes["pourcentage"] = round((stats_completes["nb_sans_ventes"] / stats_completes["nb_total"]) * 100, 2)

stats_completes = stats_completes.sort_values("nb_sans_ventes", ascending=False)

print("\nDétail par catégorie:")
print(stats_completes)
```

Nombre de produits distincts vendus : 3265/3286
 Pourcentage sans ventes : 0.64%
 Total articles non vendus : 21

Détail par catégorie:

| | categ | nb_sans_ventes | nb_total | pourcentage |
|---|-------|----------------|----------|-------------|
| 0 | 0 | 16 | 2308 | 0.69 |
| 2 | 2 | 3 | 239 | 1.26 |
| 1 | 1 | 2 | 739 | 0.27 |

▼ Caractéristique du prix par catégorie

```
df_cat0=df_merge[df_merge["categ"]==0]
print("CATEGORIE 0 :")
df_cat0["price"].describe()
```

```
df_cat1=df_merge[df_merge["categ"]==1]
print("CATEGORIE 1")
df_cat1["price"].describe()
```

```
df_cat2=df_merge[df_merge["categ"]==2]
print("CATEGORIE 2 :")
df_cat2["price"].describe()
```



```
.._-----\
# Relation entre prix et le nombre de ventes
df_agg = df_merge.groupby('id_prod').agg({
    'price': 'first',
    'categ': 'first',
    'id_prod': 'count'
}).rename(columns={'id_prod': 'sales'}).reset_index()

plt.figure(figsize=(12, 8))
sn.scatterplot(
    data=df_agg,
    x='price',
    y='sales',
    hue='categ',
    palette='viridis',
```

```
s=50,  
alpha=0.6  
)  
  
plt.title("Relation entre le prix et le nombre de ventes (par catégorie)", fontsize=14)  
plt.ylabel("Nombre de ventes", fontsize=12)  
plt.xlabel("Prix (€)", fontsize=12)  
plt.legend(title='Catégorie', bbox_to_anchor=(1.05, 1))  
plt.grid(linestyle='--', alpha=0.3)  
  
plt.tight_layout()  
plt.show()
```

```
from sklearn.cluster import KMeans

# Clusters via K-Means
X = df_agg[['price', 'sales']].values

kmeans = KMeans(n_clusters=3, random_state=42).fit(X)
df_agg['cluster'] = kmeans.labels_

sns.scatterplot(data=df_agg, x='price', y='sales', hue='cluster', palette='viridis')
plt.title("Segmentation prix/ventes (K-means)")
plt.show()
```

▼ 1) Chiffre d'affaire

▼ Chiffre d'affaire par année

```
df_merge['year'] = df_merge['date'].dt.year.astype(str)
df_ca_annuel=df_merge.groupby('year')['price'].sum().reset_index(name="ca_total")

plt.figure(figsize=(10,6))
bars=plt.bar(
```

```
df_ca_annuel["year"].astype(str),
df_ca_annuel["ca_total"],
color='#003366',
width=0.6
)
for bar in bars :
    height=bar.get_height()
    plt.annotate(
        f'{height:,.0f} €',
        xy=(bar.get_x() + bar.get_width()/2, height),
        xytext=(0, 3),
        textcoords="offset points",
        ha='center',
        va='bottom',
        fontsize=10
    )
plt.title("Chiffre d'affaire annuel", fontsize=14,pad=20,fontweight='bold')
plt.ylabel('CA (€)', fontsize=12)
plt.xlabel('Année', fontsize=12)
plt.grid(axis='y',linestyle="--",alpha=0.4)
plt.show()
```

▼ Chiffre d'affaires avec la moyenne hebdomadaire

```
df_daily = df_merge.groupby(df_merge['date'].dt.date)['price'].sum().reset_index()
df_daily['moyenne_mobile_7j'] = df_daily['price'].rolling(window=7, min_periods=1).mean()

plt.figure(figsize=(10, 5))
plt.plot(df_daily['date'], df_daily['price'], linestyle='-', label='Ventes quotidiennes', color='blue')
plt.plot(df_daily['date'], df_daily['moyenne_mobile_7j'], linestyle='-', label='Moyenne mobile 7 jours', color='red')

plt.xlabel('Date')
plt.ylabel('Total des ventes (€)')
plt.title('Évolution des ventes avec Moyenne Mobile (7 jours)')
plt.legend()
```

```
plt.xticks(rotation=45)
plt.show()
```

▼ Chiffre d'affaire MM Mois

```
df_monthly = df.merge(groupby(df.merge['date'].dt.date)['price'].sum().reset_index())
```

```
df_monthly = df_merge.groupby(df_merge['date']).agg({'price': 'sum'}).reset_index()
df_monthly['moyenne_mobile_M'] = df_monthly['price'].rolling(window=30, min_periods=1).mean()

plt.figure(figsize=(10, 5))
plt.plot(df_monthly['date'], df_monthly['price'], linestyle='-', label='Ventes quotidiennes',
plt.plot(df_monthly['date'], df_monthly['moyenne_mobile_M'], linestyle='-', label='Moyenne mol

plt.xlabel('Date')
plt.ylabel('CA (€)')
plt.title('Évolution du CA avec Moyenne Mobile (monthly)')
plt.legend()

plt.xticks(rotation=45)
plt.show()
```


Caractéristiques générales du CA (moyenne-Quartiles-Ecart type...)

[] ↪ 1 cellule masquée

▼ Chiffre d'affaire par catégorie

```
import matplotlib.pyplot as plt

df_category_sales = df_merge.groupby('categ')['price'].sum().reset_index()
df_category_sales = df_category_sales.sort_values(by='price', ascending=False)

category_colors = {
    0: '#003366',
    1: '#66b3ff',
    2: '#99ff99'
}

plt.figure(figsize=(10, 5))
bars = plt.bar(
    df_category_sales['categ'].astype(str),
    df_category_sales['price'],
    color=[category_colors[cat] for cat in df_category_sales['categ']]
)
```

```
plt.ticklabel_format(axis='y', style='plain')

for bar in bars:
    height = bar.get_height()
    plt.annotate(
        f'{height:,.0f} €',
        xy=(bar.get_x() + bar.get_width() / 2, height),
        xytext=(0, 3),
        textcoords="offset points",
        ha='center', va='bottom',
        fontsize=10
    )

plt.xlabel('Catégorie', fontsize=12)
plt.ylabel('Chiffre d\'affaires (€)', fontsize=12)
plt.title('Chiffre d\'affaires par catégorie', fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
plt.figure(figsize=(4, 4))
labels = df_category_sales['categ'].astype(str)
sizes = df_category_sales['price']
colors = [category_colors[cat] for cat in df_category_sales['categ']]

wedges, texts, autotexts = plt.pie(
    sizes,
    labels=labels,
    autopct='%1.1f%%',
    startangle=140,
    colors=colors,
    wedgeprops=dict(width=0.4),
    textprops={'color': 'grey'}
)

plt.title('Répartition du chiffre d\'affaires par catégorie (%)', fontsize=14)
plt.axis('equal')
plt.show()
```

▼ CA rapporté au nombre d'article par catégorie

```
df_ca = df_merge.groupby('categ').agg(  
    ca_total=('price', 'sum'),  
    nb_articles=('id_prod', 'nunique')  
) .reset_index()  
  
df_ca['ca_moyen_par_article'] = df_ca['ca_total'] / df_ca['nb_articles']  
  
df_ca_sorted = df_ca.sort_values('ca_moyen_par_article', ascending=False)  
  
df_ca_sorted[['categ', 'ca_total', 'nb_articles', 'ca_moyen_par_article']]
```

```
plt.figure(figsize=(8, 4))
ax = plt.subplot()

colors = ['#003366', '#66b3ff', '#99ff99']
bars = ax.barh(
    y=df_ca['categ'],
    width=df_ca['ca_moyen_par_article'],
    color=colors,
    height=0.6
)

for bar in bars:
    width = bar.get_width()
    ax.text(
        width + 500,
        bar.get_y() + bar.get_height()/2,
        f'{width:,.0f} €',
        va='center',
        ha='left',
        color='black',
        fontsize=12
    )

ax.set_yticks(df_ca['categ'])
```

```
ax.tick_params(axis='y', length=0)
ax.set_ylabel('')

plt.title('CA rapporté au nombre d article par catégorie', pad=20, fontweight='bold', fontsize=14)
plt.xlim(0, max(df_ca['ca_moyen_par_article']) * 1.3)
plt.tight_layout()

plt.show()
```

▼ Répartition Hebdomadaire du CA

```
df_merge['day_of_week'] = df_merge['date'].dt.day_name()
df_ca_per_day = df_merge.groupby('day_of_week')['price'].sum().reset_index()
```

```
df_ca_per_day = df_merge.groupby('day_of_week')['price'].sum().reset_index()
days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df_ca_per_day['day_of_week'] = pd.Categorical(df_ca_per_day['day_of_week'],
                                              categories=days_order,
                                              ordered=True)

df_ca_per_day = df_ca_per_day.sort_values('day_of_week')

plt.figure(figsize=(12, 6))
sn.set_theme(style="whitegrid")
palette = sn.color_palette("Blues_d", n_colors=7)

ax = sn.barplot(x='day_of_week',
               y='price',
               data=df_ca_per_day,
               palette=palette)

plt.title('Distribution Hebdomadaire du Chiffre d\'Affaires\n',
         fontsize=14,
         fontweight='bold',
         pad=20)
plt.xlabel('Jour de la Semaine', fontsize=12)
plt.ylabel('Total du Chiffre d\'Affaires (CA)', fontsize=12)
plt.xticks(fontsize=10, rotation=45)
plt.yticks(fontsize=10)

for p in ax.patches:
    height = p.get_height()
    y_min = plt.ylim()[0]
    y_offset = 0.02 * (df_ca_per_day['price'].max() - y_min)
    plt.text(p.get_x() + p.get_width()/2,
             height + y_offset,
             '{:1.0f}'.format(height),
```

```
        ha='center',
        va='bottom',
        fontsize=10)

mean_val = df_ca_per_day['price'].mean()
plt.axhline(mean_val,
            color='red',
            linestyle='--',
            linewidth=1)
plt.text(6.5,
        mean_val + max(df_ca_per_day['price'])*0.02,
        f'Moyenne: {mean_val:.1f}',
        color='red',
        fontsize=10)

plt.ylim(1650000, df_ca_per_day['price'].max() * 1.05)

sn.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```


▼ Distribution du CA en fonction des heures

```
df_merge['hour'] = df_merge['date'].dt.hour
df_ca_per_hour = df_merge.groupby('hour')['price'].sum().reset_index()

plt.figure(figsize=(12, 6))
sn.set_theme(style="whitegrid")
palette = sn.color_palette("Blues_d", n_colors=len(df_ca_per_hour))
```

```
ax = sns.barplot(x= hour ,
                  y='price',
                  data=df_ca_per_hour,
                  palette=palette)

plt.title('Distribution du Chiffre d\'Affaires par Heure de Transaction\n',
          fontsize=14,
          fontweight='bold',
          pad=20)
plt.xlabel('Heure de la Transaction', fontsize=12)
plt.ylabel('Chiffre d\'Affaires (CA)', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    height = p.get_height()
    y_min = plt.ylim()[0]
    y_offset = 0.02 * (df_ca_per_hour['price'].max() - y_min)
    plt.text(p.get_x() + p.get_width()/2,
             height + y_offset,
             '{:1.0f}'.format(height),
             ha='center',
             va='bottom',
             fontsize=10)

mean_val = df_ca_per_hour['price'].mean()
plt.axhline(mean_val,
            color='red',
            linestyle='--',
            linewidth=1)

plt.text(
```

```
-----\
    x=df_ca_per_hour['hour'].max() + 0.5,
    y=mean_val,
    s=f'Moyenne: {mean_val:.1f}',
    color='red',
    fontsize=10
)

plt.ylim(460000, df_ca_per_hour['price'].max() * 1.05)

sn.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```

2) Clients Distincts

[] ↳ 3 cellules masquées

▼ 3) Analyse des Ventes

▼ Ventes Général

```
df_sells_per_products=df_merge.groupby('id_prod')['id_prod'].count().reset_index(name="nb_vent")
df_sells_per_products.describe()
```

▼ Nombre de vente par année

```
df_products_per_year = df_merge.groupby('year')['id_prod'].count().reset_index()

plt.figure(figsize=(10, 5))
bars = plt.bar(df_products_per_year['year'], df_products_per_year['id_prod'], color='lightgreen')

plt.xlabel('Année', fontsize=12)
plt.ylabel('Nombre total de produits vendus', fontsize=12)
plt.title('Nombre total de produits vendus par année', fontsize=14)

for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        height + max(df_products_per_year['id_prod']) * 0.01,
        f'{height:,}',
        ha='center',
        va='bottom',
        fontsize=11,
        color='black'
```

```
)
```

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

▼ Vente par catégorie de produits

```
import matplotlib.pyplot as plt

df_products_per_category_month = df_merge.groupby(['year_month', 'categ'])['id_prod'].count()

category_colors = {
    0: '#003366',
    1: '#66b3ff',
    2: '#99ff99'
}

plt.figure(figsize=(10, 5))

for category in df_products_per_category_month['categ'].unique():
    category_data = df_products_per_category_month[df_products_per_category_month['categ'] ==
    category]
    plt.plot(
        category_data['year_month'].astype(str),
        category_data['id_prod'],
        label=f'Catégorie {category}',
        color=category_colors[category]
    )

plt.xlabel('Mois', fontsize=12)
plt.ylabel('Nombre de produits vendus', fontsize=12)
plt.title('Nombre de produits vendus par catégorie, par mois', fontsize=14)
plt.legend()

plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt
import pandas as pd

df_products_per_category = df_merge.groupby('categ')['id_prod'].count().reset_index()
df_products_per_category = df_products_per_category.sort_values(by='id_prod', ascending=False)

category_colors = {
    0: '#003366',
    1: '#66b3ff',
```



```
2: '#99ff99'
}

plt.figure(figsize=(12, 6))
bars = plt.bar(
    df_products_per_category['categ'].astype(str),
    df_products_per_category['id_prod'],
    color=[category_colors[cat] for cat in df_products_per_category['categ']]
)

for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        height + 100,
        f'{int(height)}',
        ha='center', va='bottom',
        fontsize=9, color='black'
    )

plt.xlabel('Catégorie', fontsize=12)
plt.ylabel('Nombre total de produits vendus', fontsize=12)
plt.title('Nombre total de produits vendus par catégorie', fontsize=14)

plt.xticks(rotation=45)

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

```
import matplotlib.pyplot as plt
import pandas as pd

df_products_per_category = df_merge.groupby('categ')['id_prod'].count().reset_index()
```

```
df_products_per_category = df_products_per_category.sort_values(by='id_prod', ascending=False)

category_colors = {
    0: '#003366',
    1: '#66b3ff',
    2: '#99ff99' }

plt.figure(figsize=(8, 8))
plt.pie(
    df_products_per_category['id_prod'],
    labels=df_products_per_category['categ'],
    autopct='%1.1f%%',

    textprops={'color': 'gray'},
    colors=[category_colors[c] for c in df_products_per_category['categ']],
    wedgeprops={'edgecolor': 'white', 'width': 0.4} # → effet "donut"
)

plt.title('Répartition du Nombre de Produits Vendus par Catégorie', fontsize=14)
plt.gca().set_aspect('equal')
plt.show()
```

Nombre de vente rapporté aux nombres d'articles par catégories

[] ↳ 2 cellules masquées

▼ Saisonnalité des ventes (Mois)

```
df_merge["month"]=df_merge["date"].dt.month
```

```
df_sales_per_month = df_merge.groupby('month')['id_prod'].count().reset_index()

df_sales_per_month['month_name'] = df_sales_per_month['month'].apply(lambda x: pd.to_datetime(x).strftime('%B %Y'))

df_sales_per_month = df_sales_per_month.sort_values('month')

plt.figure(figsize=(12, 6))
bars = plt.bar(
    df_sales_per_month['month_name'],
    df_sales_per_month['id_prod'],
    color='skyblue'
)

for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        height + 100,
        f'{int(height)}',
        ha='center', va='bottom',
        fontsize=9, color='black'
    )

mean_sales = df_sales_per_month['id_prod'].mean()
plt.axhline(y=mean_sales, color='red', linestyle='--', linewidth=1.5)
plt.text(11,
    mean_sales + 1000,
    f'Moyenne: {mean_sales:.0f}',
    color='red',
    fontsize=10,
    fontweight='bold')
```

```
)

plt.xlabel('Mois', fontsize=12)
plt.ylabel('Nombre total de ventes', fontsize=12)
plt.title('Saisonnalité des ventes - Nombre de ventes par mois', fontsize=14)
plt.xticks(rotation=45)
plt.ylim(45000, df_sales_per_month['id_prod'].max() + 5000)

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

Saisonnalité des ventes par catégorie

```
df_merge['month_name'] = df_merge['date'].dt.strftime('%B')
df_products_per_category_month = df_merge.groupby(['month_name', 'categ'])['id_prod'].count()

month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'Sept
df_products_per_category_month['month_name'] = pd.Categorical(df_products_per_category_month[
df_products_per_category_month = df_products_per_category_month.sort_values('month_name')

df_month_total = df_products_per_category_month.groupby('month_name')['id_prod'].sum().reset_
df_month_total['categ'] = 'Total'

df_products_per_category_month = pd.concat([df_products_per_category_month, df_month_total])

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

categories = df_products_per_category_month['categ'].unique()

width = 0.15
x = range(len(month_order))

category_colors = {
    0: '#003366',
    1: '#66b3ff',
```

```
2: '#99ff99',
'Total': '#ff9999'
}

for i, category in enumerate(categories):
    category_data = df_products_per_category_month[df_products_per_category_month['categ'] ==
    offset = width * (i - len(categories) // 2)

    color = category_colors.get(category, '#cccccc')
    plt.bar([month_order.index(month) + offset for month in category_data['month_name']],
            category_data['id_prod'], width=width,
            label=f'Catégorie {category}' if category != 'Total' else 'Total',
            color=color, alpha=0.8)

plt.xlabel('Mois', fontsize=12)
plt.ylabel('Nombre de produits vendus', fontsize=12)
plt.title('Nombre de produits vendus par mois et par catégorie', fontsize=14)

plt.legend()

plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(x, month_order, rotation=45)
plt.tight_layout()
plt.show()
```


▼ Distribution Hebdomadaire des ventes

```
df_merge['day_of_week'] = df_merge['date'].dt.day_name()
df_sales_count_per_day = df_merge.groupby('day_of_week')['session_id'].count().reset_index()
days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df_sales_count_per_day['day_of_week'] = pd.Categorical(df_sales_count_per_day['day_of_week'], categories=days_order)
```

```
categories=days_order,
ordered=True)

df_sales_count_per_day = df_sales_count_per_day.sort_values('day_of_week')

plt.figure(figsize=(12, 6))
sn.set_theme(style="whitegrid")
palette = sn.color_palette("Blues_d", n_colors=7)

ax = sn.barplot(x='day_of_week',
                y='session_id',
                data=df_sales_count_per_day,
                palette=palette)

plt.title('Distribution Hebdomadaire des Ventes\n',
          fontsize=14,
          fontweight='bold',
          pad=20)
plt.xlabel('Jour de la Semaine', fontsize=12)
plt.ylabel('Nombre de Ventes', fontsize=12)
plt.xticks(fontsize=10, rotation=45)
plt.yticks(fontsize=10)

for p in ax.patches:
    height = p.get_height()
    y_min = plt.ylim()[0]
    y_offset = 0.02 * (df_sales_count_per_day['session_id'].max() - y_min)
    plt.text(p.get_x() + p.get_width()/2,
             height + y_offset,
             '{:1.0f}'.format(height),
             ha='center',
             va='bottom',
```

```
        fontsize=10)

mean_val = df_sales_count_per_day['session_id'].mean()
plt.axhline(mean_val,
            color='red',
            linestyle='--',
            linewidth=1)
plt.text(6.5,
        mean_val + max(df_sales_count_per_day['session_id'])*0.02,
        f'Moyenne: {mean_val:.1f}',
        color='red',
        fontsize=10)

plt.ylim(95000, df_sales_count_per_day['session_id'].max() * 1.05) # Limite inférieure à 0

sn.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```

▼ Répartition du nombre de vente en fonction des heures

```
df_merge['hour'] = df_merge['date'].dt.hour
df_sales_per_hour = df_merge.groupby('hour')['session_id'].count().reset_index()

plt.figure(figsize=(12, 6))
sn.set_theme(style="whitegrid")
palette = sn.color_palette("Blues_d", n_colors=len(df_sales_per_hour))

ax = sn.barplot(x='hour')
```

```
ax = sns.barplot(x= hour ,
                  y='session_id',
                  data=df_sales_per_hour,
                  palette=palette)

plt.title('Distribution du Nombre de Ventes par Heure de Transaction\n',
          fontsize=14,
          fontweight='bold',
          pad=20)
plt.xlabel('Heure de la Transaction', fontsize=12)
plt.ylabel('Nombre de Ventes', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    height = p.get_height()
    y_min = plt.ylim()[0]
    y_offset = 0.02 * (df_sales_per_hour['session_id'].max() - y_min)
    plt.text(p.get_x() + p.get_width()/2,
             height + y_offset,
             '{:1.0f}'.format(height),
             ha='center',
             va='bottom',
             fontsize=10)

mean_val = df_sales_per_hour['session_id'].mean()
plt.axhline(mean_val,
            color='red',
            linestyle='--',
            linewidth=1)
plt.text(23.5,
         mean_val + max(df_sales_per_hour['session_id'])*0.02,
```

```
        f'Moyenne: {mean_val:.1f}',  
        color='red',  
        fontsize=10)  
  
plt.ylim(27000, df_sales_per_hour['session_id'].max() * 1.05) # Ajuster la limite inférieure  
  
sn.despine(left=True, bottom=True)  
plt.tight_layout()  
plt.show()
```

▼ 4) Nombre de transaction

▼ Evolution Globale des transactions

```
df_transactions_per_month = df_merge.groupby('year_month')['session_id'].nunique().reset_index()  
  
plt.figure(figsize=(10, 5))  
plt.plot(  
    df_transactions_per_month['year_month'].astype(str),  
    df_transactions_per_month['session_id'],  
    marker='o',  
    color='lightcoral',
```

```
        linewidth=2
    )

plt.xlabel('Mois', fontsize=12)
plt.ylabel('Nombre de transactions', fontsize=12)
plt.title('Évolution des transactions par mois', fontsize=14)
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

▼ Nombre de transactions par catégorie

```
df_transactions_per_category = df_merge.groupby('categ')['session_id'].nunique().reset_index('')

category_colors = {
    0: '#003366',
    1: '#66b3ff',
    2: '#99ff99'
}

plt.figure(figsize=(10, 5))

bars = plt.bar(
    df_transactions_per_category['categ'].astype(str),
    df_transactions_per_category['session_id'],
    color=[category_colors[cat] for cat in df_transactions_per_category['categ']]
)

for bar in bars:
    height = bar.get_height()
    plt.annotate(
        f'{height}',
        xy=(bar.get_x() + bar.get_width() / 2, height),
        xytext=(0, 3),
        textcoords="offset points",
        ha='center', va='bottom',
        fontsize=10
    )
```



```
plt.xlabel('Catégorie', fontsize=12)
plt.ylabel('Nombre de transactions', fontsize=12)
plt.title('Nombre de transactions total par catégorie', fontsize=14)

plt.xticks(rotation=45)

plt.grid(axis='y', linestyle='--', alpha=0.7)

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

```
df_transactions_per_day = df_merge.groupby('day_of_week')['session_id'].nunique().reset_index()
days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df_transactions_per_day['day_of_week'] = pd.Categorical(df_transactions_per_day['day_of_week'],
                                                         categories=days_order,
                                                         ordered=True)
df_transactions_per_day = df_transactions_per_day.sort_values('day_of_week')
df_transactions_per_day.describe()
```

▼ Saisonnalité hebdomadaire transactions

```
plt.figure(figsize=(12, 6))
sn.set_theme(style="whitegrid")
palette = sn.color_palette("Blues_d", n_colors=7)
```

```
ax = sn.barplot(x='day_of_week',
               y='session_id',
               data=df_transactions_per_day,
               palette=palette)

plt.title('Distribution Hebdomadaire des Transactions\n',
         fontsize=14,
         fontweight='bold',
         pad=20)
plt.xlabel('Jour de la Semaine', fontsize=12)
plt.ylabel('Nombre de Transactions', fontsize=12)
plt.xticks(fontsize=10, rotation=45)
plt.yticks(fontsize=10)

for p in ax.patches:
    height = p.get_height()
    plt.text(p.get_x() + p.get_width()/2,
             height + max(df_transactions_per_day['session_id'])*0.02,
             '{:1.0f}'.format(height),
             ha='center',
             va='bottom',
             fontsize=10)

mean_val = df_transactions_per_day['session_id'].mean()
plt.axhline(mean_val,
            color='red',
            linestyle='--',
            linewidth=1)
plt.text(6.5,
        mean_val + max(df_transactions_per_day['session_id'])*0.02,
        f'Moyenne: {mean_val: 1.0f}')
```

```
    r Moyenne: {mean_val:.1f} ,  
    color='red',  
    fontsize=10)  
  
plt.ylim(45000, df_transactions_per_day['session_id'].max() * 1.05)  
  
sn.despine(left=True, bottom=True)  
plt.tight_layout()  
plt.show()
```

▼ Distribution du nombre des transactions en fonction de l'horraire

```
df_merge['hour'] = df_merge['date'].dt.hour
df_transactions_per_hour = df_merge.groupby('hour')['session_id'].count().reset_index() # Cor
df_transactions_per_hour.describe()
```

```
plt.figure(figsize=(12, 6))
plt.set_theme(style="whitegrid")
```

```
sn.set_theme(style= 'whitegrid' )
palette = sn.color_palette("Blues_d", n_colors=24)

ax = sn.barplot(x='hour',
                y='session_id',
                data=df_transactions_per_hour,
                palette=palette)

plt.title('Distribution Horaire des Transactions\n',
          fontsize=14,
          fontweight='bold',
          pad=20)
plt.xlabel('Heure de la Transaction', fontsize=12)
plt.ylabel('Nombre de Transactions', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    height = p.get_height()
    y_min = plt.ylim()[0]
    y_offset = 0.02 * (df_transactions_per_hour['session_id'].max() - y_min)
    plt.text(p.get_x() + p.get_width()/2,
             height + y_offset,
             '{:1.0f}'.format(height),
             ha='center',
             va='bottom',
             fontsize=10)

mean_val = df_transactions_per_hour['session_id'].mean()
plt.axhline(mean_val,
            color='red',
            linestyle='--',
```

```
        linewidth=1)
plt.text(23.5,
        mean_val + max(df_transactions_per_hour['session_id'])*0.02,
        f'Moyenne: {mean_val:.1f}',
        color='red',
        fontsize=10)

plt.ylim(27000, df_transactions_per_hour['session_id'].max() * 1.05) # Limite inférieure à 0

sn.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```

▼ Nombre de transactions cumulées dans le temps

```
df_daily_transactions = df_merge.groupby('date')['session_id'].nunique().reset_index()
df_daily_transactions['cumulative_transactions'] = df_daily_transactions['session_id'].cumsum()

plt.figure(figsize=(12, 6))
plt.plot(df_daily_transactions['date'], df_daily_transactions['cumulative_transactions'], color='blue')

plt.xlabel('Date', fontsize=12)
plt.ylabel('Transactions cumulées', fontsize=12)
plt.title('Évolution cumulée des transactions dans le temps', fontsize=14)
plt.grid(True)
plt.tight_layout()
plt.show()
```


▼ 5) Analyse Top Flop *produit*

```
df_merge.head()
```

▼ Top 10 références par CA

```
agg_data = df_merge.groupby(['id_prod', 'categ', 'price']).agg(  
    CA=('price', 'sum'),  
    nb_ventes=('price', 'count')  
).reset_index()
```

```
agg_data_sorted = agg_data.sort_values('CA', ascending=False)
```

```
top_10 = agg_data_sorted.head(10)  
flop_10=agg_data_sorted.tail(10)
```

```
top_10[['id_prod', 'CA', 'categ', 'price', 'nb_ventes']]
```

```
plt.style.use('seaborn-v0_8-darkgrid')
fig, ax = plt.subplots(figsize=(14, 7))

labels = [f"{str(row['id_prod'])} - {row['categ']}" for _, row in top_10.iterrows()]
ca_values = top_10['CA']
prix_moyens = top_10['price']

colors = plt.cm.Greens(np.linspace(0.4, 1, len(top_10)))

bars = ax.barh(
    labels,
    ca_values,
    color=colors,
    edgecolor='darkgreen',
    linewidth=0.7
)

for i, bar in enumerate(bars):
    width = bar.get_width()
    ax.text(width * 1.01,
            bar.get_y() + bar.get_height()/2,
            f'CA: {width:,.0f}€ | Prix moyen: {prix_moyens.iloc[i]:,.2f}€',
            va='center',
```

```
        fontsize=10,  
        fontweight='bold')  
  
ax.set_title('TOP 10 DES PRODUITS PAR CHIFFRE D\'AFFAIRES',  
            pad=20,  
            fontsize=14,  
            fontweight='bold')  
ax.set_xlabel('Chiffre d\'Affaires (€)',  
            labelpad=10,  
            fontsize=12)  
  
for spine in ['top', 'right']:  
    ax.spines[spine].set_visible(False)  
  
ax.xaxis.set_major_formatter('{x:,.0f}€')  
  
plt.tight_layout()  
plt.gca().invert_yaxis()  
plt.subplots_adjust(left=0.3)  
  
plt.show()
```

▼ Flop 10 par CA

```
flop_10[['id_prod', 'CA', 'categ', 'price', 'nb_ventes']]
```

```
plt.style.use('seaborn-v0_8-darkgrid')
fig, ax = plt.subplots(figsize=(14, 7))

labels = [f"{str(row['id_prod'])} - {row['categ']}" for _, row in flop_10.iterrows()]
ca_values = flop_10['CA']
prix_moyens = flop_10['price']

colors = plt.cm.Reds(np.linspace(0.4, 1, len(top_10)))

bars = ax.barh(
    labels,
    ca_values,
    color=colors,
    edgecolor='darkred',
    linewidth=0.7
)

for i, bar in enumerate(bars):
    width = bar.get_width()
    ax.text(width * 1.01,
            bar.get_y() + bar.get_height() / 2,
```

```
bar.get_y() + bar.get_height()/2,  
f'CA: {width:,.2f}€ | Prix moyen: {prix_moyens.iloc[i]:,.2f}€',  
va='center',  
fontsize=10,  
fontweight='bold')  
  
ax.set_title('FLOP 10 DES PRODUITS PAR CHIFFRE D\'AFFAIRES',  
            pad=20,  
            fontsize=14,  
            fontweight='bold')  
ax.set_xlabel('Chiffre d\'Affaires (€)',  
            labelpad=10,  
            fontsize=12)  
  
for spine in ['top', 'right']:  
    ax.spines[spine].set_visible(False)  
  
ax.xaxis.set_major_formatter('{x:,.2f}€')  
  
plt.tight_layout()  
plt.gca().invert_yaxis()  
plt.subplots_adjust(left=0.3)  
  
plt.show()
```

▼ Top 10 références par vente

```
vente_data_sorted = agg_data.sort_values(by='nb_ventes', ascending=False)
top_10_ventes = vente_data_sorted.head(10)
flop_10_ventes = vente_data_sorted.tail(20)
```

```
top_10_ventes
```


top_10_ventes

Étapes suivantes :

[Générer du code avec top_10_ventes](#)

[Afficher les graphiques recommandés](#)

[New interactive sheet](#)

```
plt.style.use('seaborn-v0_8-darkgrid')
fig, ax = plt.subplots(figsize=(14, 7))

labels = [f"{str(row['id_prod'])} - {row['categ']}" for _, row in top_10_ventes.iterrows()]
ventes_values = top_10_ventes['nb_ventes']
prix = top_10_ventes['price']

colors = plt.cm.Greens(np.linspace(0.4, 1, len(top_10)))

bars = ax.barh(
    labels
```

```
labels,
ventes_values,
color=colors,
edgecolor='darkgreen',
linewidth=0.7
)

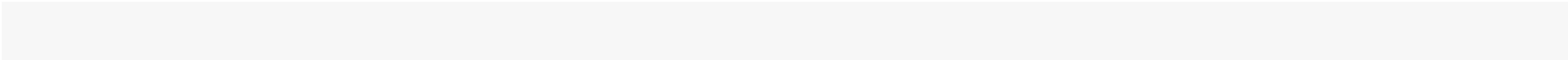
for i, bar in enumerate(bars):
    width = bar.get_width()
    ax.text(width * 1.01,
            bar.get_y() + bar.get_height()/2,
            f'Nombre ventes: {width} | Prix : {prix.iloc[i]:,.2f}€',
            va='center',
            fontsize=10,
            fontweight='bold')

ax.set_title('TOP 10 DES PRODUITS PAR NOMBRE DE VENTES',
            pad=20,
            fontsize=14,
            fontweight='bold')
ax.set_xlabel('Nombre de ventes ',
            labelpad=10,
            fontsize=12)

for spine in ['top', 'right']:
    ax.spines[spine].set_visible(False)

plt.tight_layout()
plt.gca().invert_yaxis()
plt.subplots_adjust(left=0.3)

plt.show()
```



▼ Flop 10 références par vente

```
flop_10_ventes
```

Étapes suivantes :

[Générer du code avec flop_10_ventes](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

```
plt.style.use('seaborn-v0_8-darkgrid')
fig, ax = plt.subplots(figsize=(14, 7))

labels = [f"{str(row['id_prod'])} - {row['categ']}" for _, row in flop_10_ventes.iterrows()]
ventes_values = flop_10_ventes['nb_ventes']
prix = flop_10_ventes['price']

colors = plt.cm.Reds(np.linspace(0.4, 1, len(top_10)))

bars = ax.barh(
    labels,
    ventes_values,
    color=colors,
    edgecolor='darkred',
    linewidth=0.7
)

for i, bar in enumerate(bars):
    width = bar.get_width()
    ax.text(width * 1.01,
            bar.get_y() + bar.get_height()/2,
            f'Nombre ventes: {width} | Prix : {prix.iloc[i]:.2f}€',
            va='center',
            fontsize=10,
            fontweight='bold')
```

```
fontweight= bold )

ax.set_title('FLOP 20 DES PRODUITS PAR NOMBRE DE VENTES',
            pad=20,
            fontsize=14,
            fontweight='bold')
ax.set_xlabel('Nombre de ventes ',
            labelpad=10,
            fontsize=12)

for spine in ['top', 'right']:
    ax.spines[spine].set_visible(False)

plt.tight_layout()
plt.gca().invert_yaxis()
plt.subplots_adjust(left=0.3)

plt.show()
```

▼ Courbe de Lorentz CA produit

```
def lorentz_curve(data, value_col):  
    data_sorted = data.sort_values(by=value_col, ascending=False)  
  
    total_value = data_sorted[value_col].sum()  
    cumulative_value = data_sorted[value_col].cumsum()  
    cumulative_percentage_value = cumulative_value / total_value * 100  
  
    num_products = data_sorted.shape[0]  
    cumulative_percentage_products = np.linspace(0, 100, num_products)
```

```
    return cumulative_percentage_products, cumulative_percentage_value

x_CA, y_CA = lorentz_curve(agg_data, 'CA')

x_sales, y_sales = lorentz_curve(agg_data, 'nb_ventes')

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(x_CA, y_CA, label="CA", color='blue', lw=2)
ax.plot([0, 100], [0, 100], '--', color='red', label="Ligne d'équité (50% produits = 50% CA)")
ax.set_xlabel('Pourcentage de Produits')
ax.set_ylabel('Pourcentage du CA')
ax.set_title('Courbe de Lorentz - Distribution du Chiffre d\'Affaires')
ax.legend()
plt.tight_layout()
plt.show()

threshold_CA = np.argmax(y_CA >= 80)
print(f"80% du CA est atteint à partir de {round(x_CA[threshold_CA],2)}% des produits.")
```


▼ Courbe de Lorentz Vente Produit

```
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(x_sales, y_sales, label="Ventes", color='green', lw=2)
ax.plot([0, 100], [0, 100], '--', color='red', label="Ligne d'équité (50% produits = 50% Ventes)")
ax.set_xlabel('Pourcentage de Produits')
ax.set_ylabel('Pourcentage des Ventes')
ax.set_title('Courbe de Lorentz - Distribution des Ventes')
ax.legend()
plt.tight_layout()
plt.show()

threshold_sales = np.argmax(y_sales >= 80)
print(f"80% des ventes sont atteintes à partir de {round(x_sales[threshold_sales],2)}% des produits")
```

- ▼ Quel est la répartition des catégories des tops produits permettant d'atteindre 80% du CA?

```
df_CA_cat = df_merge.groupby(["id_prod", "categ"])["price"].sum().reset_index(name="CA")
```

```
df_CA_cat = df_CA_cat.sort_values(by='CA', ascending=False).reset_index(drop=True)

df_CA_cat["CA_cum"] = df_CA_cat["CA"].cumsum()
df_CA_cat["CA_cumule_pct"] = df_CA_cat["CA_cum"] / df_CA_cat["CA"].sum()

top_articles_80_CA = df_CA_cat[df_CA_cat["CA_cumule_pct"] <= 0.8]

ca_par_cat = top_articles_80_CA.groupby("categ")["CA"].sum()
total_CA_top80 = top_articles_80_CA["CA"].sum()
top_cat_80_CA_count = top_articles_80_CA["categ"].value_counts()
top_cat_80_CA_percent = top_articles_80_CA["categ"].value_counts(normalize=True)
CA_cat_percent = (ca_par_cat / total_CA_top80 * 100).round(2)

df_top_cat_CA = pd.DataFrame({
    "Nombre d'articles": top_cat_80_CA_count,
    "Proportion d'articles (%)": (top_cat_80_CA_percent * 100).round(2),
    "Part du CA cumulé (%)": CA_cat_percent
})

df_top_cat_CA
```

Étapes suivantes :

[Générer du code avec df_top_cat_CA](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

```
plt.figure(figsize=(15,6))
sn.boxplot(x=top_articles_80_CA["CA"])

plt.title('Répartition des CA des produits(80% du CA cumulées)')
plt.xlabel("CA par produit")
plt.tight_layout()
plt.show()
```

```
top_articles_80_CA["CA"].describe()
```

▼ Caractéristiques prix top produit permettant d'atteindre 80% du CA

```
df_CA_80=df_merge.groupby("id_prod").agg(  
    CA=("price","sum"),  
    price=("price","mean")  
)  
df_CA_80.sort_values(by="price", ascending=False)
```

```
df_CA_80["CA_cum"]=df_CA_80["price"].cumsum()  
df_CA_80["CA_cum_percent"]=df_CA_80["CA_cum"]/df_CA_80["CA"]  
df_CA_80=df_CA_80[df_CA_80["CA_cum_percent"]<=0.8]  
df_CA_80["price"].describe()
```

▼ Répartition par catégorie Top produit qui permet d'atteindre 80% des ventes

```
df_80_ventes_by_cat = df_merge.groupby(["id_prod", "categ"]).agg(  
    nb_ventes=("id_prod", "count")  
)  
  
df_80_ventes_by_cat = df_80_ventes_by_cat.sort_values(by="nb_ventes", ascending=False).reset_:
```

```
total_ventes = df_80_ventes_by_cat["nb_ventes"].sum()
df_80_ventes_by_cat["vente_cum"] = df_80_ventes_by_cat["nb_ventes"].cumsum()
df_80_ventes_by_cat["vente_cum_pct"] = df_80_ventes_by_cat["vente_cum"] / total_ventes

df_80_ventes_by_cat = df_80_ventes_by_cat[df_80_ventes_by_cat["vente_cum_pct"] < 0.8]

nb_articles_par_cat = df_80_ventes_by_cat["categ"].value_counts()

prop_articles_par_cat = df_80_ventes_by_cat["categ"].value_counts(normalize=True) * 100

df_top_cat_ventes = pd.DataFrame({
    "Nombre d'articles": nb_articles_par_cat,
    "Proportion d'articles (%)": prop_articles_par_cat
})
df_top_cat_ventes
```

Étapes suivantes :

[Générer du code avec df_top_cat_ventes](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

Boite à Moustache répartition du TOP qui a généré 80% des ventes

[] ↪ 1 cellule masquée

▼ Caractéristiques du prix moyen des articles permettant d'attendre 80% des ventes

```
df_ventes = df_merge.groupby('id_prod').agg(  
    nb_ventes=('id_prod', 'count'),  
    price=('price', 'mean')  
) .reset_index()  
  
df_ventes = df_ventes.sort_values('nb_ventes', ascending=False)  
  
df_ventes['ventes_cum'] = df_ventes['nb_ventes'].cumsum()  
total_ventes = df_ventes['nb_ventes'].sum()  
df_ventes['pct_cumule'] = 100 * df_ventes['ventes_cum'] / total_ventes  
  
df_80_ventes = df_ventes[df_ventes['pct_cumule'] <= 80]  
  
df_80_ventes['price'].describe()
```


▼ **MOST IMPORTANT PRODUCTS** (Produits appartenant dans le TOP génération 80% CA et Ventes)

```
df_CA_80.head()
```

Étapes suivantes :

[Générer du code avec df_CA_80](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

```
df_most_important_products = df_80_ventes.merge(df_CA_80, how='inner', on='id_prod', suffixes=
df_cat=df_merge[["id_prod","categ"]].drop_duplicates()
df_most_important_products.describe()
```

```
df_most_important_products=df_most_important_products.merge(df_cat,how="inner", on="id_prod")
```

```
df_most_important_products.describe()
```

```
df_most_important_products.drop(columns=['price_left'], inplace=True)  
df_most_important_products.rename(columns={'price_right': 'price'}, inplace=True)
```

```
df_most_important_products.head()
```

Étapes
suivantes :

[Générer du code avec df_most_important_products](#)
[Afficher les graphiques recommandés](#)
[New interactive sheet](#)

```
CA_total = df_most_important_products["CA"].sum()
VENTES_total = df_most_important_products["nb_ventes"].sum()

CA_total_librairie = df_ca["ca_total"].sum()
VENTES_total_librairie = df_ventes["nb_ventes"].sum()

pourcentage_CA = round((CA_total / CA_total_librairie) * 100, 2)
pourcentage_ventes = round((VENTES_total / VENTES_total_librairie) * 100, 2)
total_best_products=df_most_important_products["id_prod"].count()
total_products=len(df_merge["id_prod"].unique())
print(f"Les produits les plus importants génèrent {CA_total} € de chiffre d'affaires, soit {pourcentage_CA}% du total des ventes")
print(f"Ils représentent aussi {VENTES_total} ventes, soit {pourcentage_ventes}% du total des ventes")
print(f"Ces produits sont au nombre de {total_best_products}/{total_products} soit {round((total_best_products / total_products) * 100, 2)}%")
```

Les produits les plus importants génèrent 3377548.32 € de chiffre d'affaires, soit 28.08% du total.
Ils représentent aussi 216691 ventes, soit 31.52% du total des ventes.
Ces produits sont au nombre de 299/3265 soit 9.16%.

```
df_cat_article_count = df_most_important_products["categ"].value_counts().reset_index(name="nb_articles")
df_cat_article_count.columns = ["categ", "nb_articles"]
df_cat_article_count
```

Étapes suivantes :

[Générer du code avec df_cat_article_count](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

```
df_cat_article_count["pct"] = (df_cat_article_count["nb_articles"] / df_cat_article_count["nb_
category_colors = {
    0: '#003366',
    1: '#66b3ff',
    2: '#99ff99'
}

labels = [
    f"Categorie {row.categ}\n{row.nb_articles} articles\n{row.pct}%"
    for row in df_cat_article_count.itertuples()
]

colors = [category_colors[c] for c in df_cat_article_count["categ"]]
sizes = df_cat_article_count["nb_articles"]

fig, ax = plt.subplots(figsize=(8, 6))
wedges, _ = ax.pie(sizes, colors=colors, startangle=90, wedgeprops=dict(width=0.4))

plt.legend(wedges, labels, title="Catégories", loc="center left", bbox_to_anchor=(1, 0.5))

centre_circle = plt.Circle((0, 0), 0.70, fc='white')
ax.add_artist(centre_circle)
```

```
ax.add_artist(circle_circle)

ax.set_title("Répartition des articles par catégorie")
plt.tight_layout()
plt.show()
```

```
df_most_important_products["price"].describe()
```

```
df_most_important_products["nb_ventes"].describe()
```

```
df_most_important_products["CA"].describe()
```

```
df_most_important_products["id_prod"].tolist()
```

```
['1_369',  
 '1_417',  
 '1_414',  
 '1_498',  
 '1_406',  
 '1_395',  
 '1_385',  
 '1_383',  
 '1_389',  
 '1_378',  
 '1_379',
```

```
'_1_431',  
'1_366',  
'0_1422',  
'0_1431',  
'0_1425',  
'0_1432',  
'0_1411',  
'0_0',  
'0_1424',  
'0_1441',  
'0_1434',  
'0_1430',  
'0_1445',  
'0_1416',  
'0_1438',  
'0_1419',  
'0_1417',  
'0_1410',  
'0_1476',  
'0_1446',  
'0_1474',  
'0_1429',  
'0_1435',  
'0_1420',  
'0_1473',  
'0_1471',  
'0_1426',  
'0_1421',  
'0_1414',  
'0_1412',  
'0_1475',  
'0_1479',  
'0_1358',  
'0_1457',  
'0_1448',  
'0_1407',  
'0_1451',  
'0_1366',  
'0_1464',  
'0_1472',  
'0_1453',  
..
```



```
'0_1461',  
'0_1357',  
'0_1418',  
'0_1350',  
'0_1404',  
'0_1470',
```

▼ 6) Analyse par profil client(BtoB & Particulier)

```
df_merge.head()
```

Les clients BtoB désignent des entreprises qui achètent des produits pour les besoins de leur activité professionnelle. Ces clients ont généralement un volume et une fréquence d'achat plus élevés que les clients particuliers.

Dans le cadre de cette analyse, nous faisons l'hypothèse que la majorité des clients de notre librairie sont des clients retail (particuliers).

Par conséquent, nous considérons que les valeurs extrêmes en termes de fréquence et de volume d'achat correspondent probablement à des clients BtoB.

```
clients = df_merge.groupby("client_id").agg(  
    CA=('price', 'sum'),  
    nb_vente=('client_id', 'count')
```

```

).reset_index()

Q1_vente=clients["nb_vente"].quantile(0.25)
Q3_vente=clients["nb_vente"].quantile(0.75)
IQR_vente=Q3_vente-Q1_vente

Q1_CA=clients["CA"].quantile(0.25)
Q3_CA=clients["CA"].quantile(0.75)
IQR_CA=Q3_CA-Q1_CA
outliers=clients[(clients["CA"] > Q3_CA + 1.5 * IQR_CA) &
                  (clients["nb_vente"] > Q3_vente + 1.5 * IQR_vente)]

clients["type"] = "Retail"
clients.loc[
    (clients["CA"] > Q3_CA + 1.5 * IQR_CA) & (clients["nb_vente"] > Q3_vente + 1.5 * IQR_vente)
    , "type"
] = "BtoB"

print(f"Les clients BtoB sont aux nombre de {len(outliers)}. Ils ont un volume d'achat et un CA")
print(outliers)

```

Les clients BtoB sont aux nombre de 199. Ils ont un volume d'achat et un CA supérieur aux autres clients

| | client_id | CA | nb_vente |
|------|-----------|---------|----------|
| 9 | c_1006 | 4029.97 | 299 |
| 30 | c_1025 | 3727.89 | 261 |
| 84 | c_1074 | 4152.27 | 319 |
| 113 | c_110 | 4156.41 | 309 |
| 266 | c_1239 | 3697.69 | 272 |
| ... | ... | ... | ... |
| 8397 | c_8574 | 4125.66 | 310 |
| 8427 | c_8600 | 4752.55 | 319 |
| 8454 | c_867 | 3727.42 | 283 |
| 8469 | c_880 | 4897.19 | 365 |

```
8560      c_963  4482.09      359
```

```
[199 rows x 3 columns]
```

```
clients.head()
```

Étapes suivantes :

[Générer du code avec clients](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

▼ Répartition des clients en fonction de leur type

```
nb_clients = clients['type'].value_counts()
pourcentage_clients = (nb_clients / nb_clients.sum() * 100).round(2)
print(f"On dénombre au total {nb_clients.sum()} clients.")
print(f"Les particuliers sont {nb_clients['Retail']}.")
print(f"Les BtoB sont {nb_clients['BtoB']}.")
```

```
On dénombre au total 8600 clients.
Les particuliers sont 8401.
Les BtoB sont 199.
```

```
plt.figure(figsize=(6,6))
nb_clients.plot.pie(
```

```
autopct='%1.1t%%',  
startangle=90,  
colors=['#4C72B0', '#55A868'], # Couleurs fixes pour Retail et BtoB  
labels=['Retail', 'BtoB'],  
wedgeprops={'edgecolor': 'black'}  
)  
plt.title("Répartition des clients : Retail vs BtoB")  
plt.ylabel("")  
plt.show()
```

▼ Chiffre d'affaire

```
ca_par_type = clients.groupby('type')['CA'].sum()
print("Chiffre d'affaires total par type de client :")
print(ca_par_type)
```

```
Chiffre d'affaires total par type de client :
type
BtoB      1700062.87
Retail    10327600.23
Name: CA, dtype: float64
```

```
ca_par_type = ca_par_type.reindex(['Retail', 'BtoB'])

plt.figure(figsize=(6,6))
ca_par_type.plot.pie(
    autopct='%1.1f%%',
    startangle=90,
    colors=['#4C72B0', '#55A868'],
    labels=['Retail', 'BtoB'],
    wedgeprops={'edgecolor': 'black'}
)
plt.title("Part du CA total : Retail vs BtoB")
plt.ylabel("")
plt.show()
```

```
custom_palette = {
    'Retail': '#4C72B0',
    'BtoB': '#55A868'
}

df_merge = df_merge.merge(clients[['client_id', 'type']], on='client_id', how='left')
ca_categorie = df_merge.groupby(['type', 'categ'])['price'].sum().reset_index()

plt.figure(figsize=(10,6))
ax = sn.barplot(data=ca_categorie, x='categ', y='price', hue='type', palette=custom_palette)
plt.title("Répartition du CA par catégorie de produit (Retail vs BtoB)")
plt.ylabel("Chiffre d'affaires")
plt.xticks(rotation=45)
```

```
patches = ax.patches

for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.text(p.get_x() + p.get_width()/2,
                height + ca_categorie['price'].max() * 0.01,
                f'{height:,.0f}',
                ha='center',
                va='bottom',
                fontsize=9)

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

```
import matplotlib.pyplot as plt

category_colors = {
    0: '#003366',
    1: '#66b3ff',
    2: '#99ff99'
}

ca_categorie_retail = ca_categorie[ca_categorie['type'] == 'Retail']
ca_categorie_b2b = ca_categorie[ca_categorie['type'] == 'BtoB']
```

```
plt.figure(figsize=(8, 8))
plt.pie(
    ca_categorie_retail['price'],
    labels=ca_categorie_retail['categ'],
    autopct='%1.1f%%',
    textprops={'color': 'gray'},
    startangle=90,
    colors=[category_colors[c] for c in ca_categorie_retail['categ']],
    wedgeprops={'edgecolor': 'white', 'width': 0.4}
)
plt.title("Répartition du CA par catégorie - Retail")
plt.gca().set_aspect('equal')
plt.show()

plt.figure(figsize=(8, 8))
plt.pie(
    ca_categorie_b2b['price'],
    labels=ca_categorie_b2b['categ'],
    autopct='%1.1f%%',
    textprops={'color': 'gray'},
    startangle=90,
    colors=[category_colors[c] for c in ca_categorie_b2b['categ']],
    wedgeprops={'edgecolor': 'white', 'width': 0.4}
)
plt.title("Répartition du CA par catégorie - BtoB")
plt.gca().set_aspect('equal')
plt.show()
```


▼ Ventes

```
ventes_par_type = df_merge.groupby('type')['session_id'].count()

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

```
ventes_par_type.plot.pie(  
    autopct='%1.1f%%',  
    startangle=90,  
    colors=['#4C72B0', '#55A868'], # Bleu Retail, Vert BtoB  
    labels=['Retail', 'BtoB'],  
    wedgeprops={'edgecolor': 'black'}  
)  
plt.title("Part des ventes : Retail vs BtoB")  
plt.ylabel("")  
plt.show()
```

```
# Définir la palette personnalisée pour cohérence
custom_palette = {
    'Retail': '#4C72B0', # Bleu
    'BtoB': '#55A868'    # Vert
}

# Calcul du nombre de ventes par catégorie et par type de client
ventes_categorie = df_merge.groupby(['type', 'categ'])['session_id'].count().reset_index()

# Affichage du graphique
plt.figure(figsize=(10,6))
ax = sns.barplot(data=ventes_categorie, x='categ', y='session_id', hue='type', palette=custom_
plt.title("Répartition du Nombre de Ventes par Catégorie de Produit (Retail vs BtoB)")
plt.ylabel("Nombre de Ventes")
plt.xlabel("Catégorie")
plt.xticks(rotation=45)

# → Ajout des chiffres sur les barres
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.text(p.get_x() + p.get_width()/2,
                height + ventes_categorie['session_id'].max() * 0.01,
                f'{height:,}', # Format avec séparateur de milliers
                ha='center',
                va='bottom',
                fontsize=9)
```

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

```
# 1 Palette de couleurs personnalisée (identique)
category_colors = {
    0: '#003366', # Bleu foncé
    1: '#66b3ff', # Bleu clair
    2: '#99ff99'  # Vert clair
}

# 3 Séparation Retail et BtoB
ventes_categorie_retail = ventes_categorie[ventes_categorie['type'] == 'Retail']
ventes_categorie_b2b = ventes_categorie[ventes_categorie['type'] == 'BtoB']

# 4 Graphique en donut pour Retail
plt.figure(figsize=(8, 8))
plt.pie(
    ventes_categorie_retail['session_id'],
    labels=ventes_categorie_retail['categ'],
    autopct='%1.1f%%',
    textprops={'color': 'gray'},
    startangle=90,
    colors=[category_colors[c] for c in ventes_categorie_retail['categ']],
    wedgeprops={'edgecolor': 'white', 'width': 0.4}
)
plt.title("Répartition du Nombre de Ventes par Catégorie - Retail")
plt.gca().set_aspect('equal')
plt.show()

# 5 Graphique en donut pour BtoB
plt.figure(figsize=(8, 8))
plt.pie(
    ventes_categorie_b2b['session_id'],
```

```
    labels=ventes_categorie_b2b['categ'],
    autopct='%1.1f%%',
    textprops={'color': 'gray'},
    startangle=90,
    colors=[category_colors[c] for c in ventes_categorie_b2b['categ']],
    wedgeprops={'edgecolor': 'white', 'width': 0.4}
)
plt.title("Répartition du Nombre de Ventes par Catégorie - BtoB")
plt.gca().set_aspect('equal')
plt.show()
```


▼ 7) Analyse profil client

▼ Calcul temps moyen entre 2 achats clients

```
clients_valides = df_merge.groupby('client_id').filter(lambda x: x['session_id'].nunique() >= 2)

temps_moyen = clients_valides.sort_values(['client_id', 'date'])\
    .groupby('client_id')['date']\
    .apply(lambda x: x.diff().mean().days)

print(f"Temps moyen entre 2 achats : {temps_moyen.mean():.0f} jours")
print(f"Mediane entre 2 achats : {temps_moyen.median():.0f} jours")
print(f"Ecart Type   : {temps_moyen.std():2f}")
```

```
Temps moyen entre 2 achats : 20 jours
Mediane entre 2 achats : 12 jours
Ecart Type   : 27.910580
```

```
temps_par_client = clients_valides.groupby('client_id')['date']\
    .apply(lambda x: x.diff().mean().days)

plt.hist(temps_par_client.dropna(), bins=30, edgecolor='k')
```

```
plt.axvline(temps_par_client.mean(), color='red', label=f'Moyenne: {temps_par_client.mean():.0f}')
plt.xlabel("Jours entre 2 achats")
plt.ylabel("Nombre de clients")
plt.legend()
plt.show()
```

▼ Courbe de lorentz: Client par rapport au CA

```
from matplotlib.ticker import PercentFormatter
```

```
ca_par_client = df_merge.groupby('client_id')['price'].sum().sort_values(ascending=False).reset_index()
ca_par_client['ca_cumul'] = ca_par_client['price'].cumsum()
ca_par_client['%_ca_cumul'] = ca_par_client['ca_cumul'] / ca_par_client['price'].sum() * 100
ca_par_client['%_clients_cumul'] = (ca_par_client.index + 1) / len(ca_par_client) * 100

plt.figure(figsize=(10, 6))
plt.plot([0, 100], [0, 100], 'k--', label='Égalité parfaite') # Ligne de référence
plt.plot(ca_par_client['%_clients_cumul'], ca_par_client['%_ca_cumul'], 'b-', lw=2, label='Concentration du CA')

AUC_lorenz = np.trapz(ca_par_client['%_ca_cumul'], ca_par_client['%_clients_cumul'])
AUC_perfect = 5000
gini = (AUC_perfect - AUC_lorenz) / AUC_perfect

plt.fill_between(ca_par_client['%_clients_cumul'], ca_par_client['%_ca_cumul'], alpha=0.2)
plt.title(f'Concentration du CA parmi les clients\nIndice de Gini = {gini:.2f}', pad=20)
plt.xlabel('% Cumulé des clients', labelpad=10)
plt.ylabel('% Cumulé du CA', labelpad=10)
plt.legend(loc='upper left')
plt.grid(True, linestyle='--', alpha=0.3)
plt.gca().yaxis.set_major_formatter(PercentFormatter())
plt.gca().xaxis.set_major_formatter(PercentFormatter())
plt.tight_layout()
plt.show()
```

- ✓ 8) Segmentation des clients par RFM(Récence,Fréquence,Montant)
- ✓ Calcul du score RFM par client

```
now = df_merge["date"].max()
rfm = df_merge.groupby("client_id").agg({
    "date": lambda x: (now - x.max()).days,
    "session_id": "nunique",
    "price": "sum"
```

```
}).reset_index()

rfm.columns = ["client_id", "Recence", "Frequence", "Montant"]

rfm["R_score"] = pd.qcut(rfm["Recence"], 5, labels=[5,4,3,2,1])
rfm["F_score"] = pd.qcut(rfm["Frequence"].rank(method="first"), 5, labels=[1,2,3,4,5])
rfm["M_score"] = pd.qcut(rfm["Montant"], 5, labels=[1,2,3,4,5])

rfm["RFM_score"] = rfm["R_score"].astype(str) + rfm["F_score"].astype(str) + rfm["M_score"].a:

rfm.head()
```

Étapes suivantes :

[Générer du code avec rfm](#)[Afficher les graphiques recommandés](#)[New interactive sheet](#)

```
def segmenter(rfm_score):
    if rfm_score == "555":
        return "Champions"
    elif rfm_score[0] == "5":
        return "Clients récents"
    elif rfm_score[1] == "5":
        return "Clients fréquents"
```

```
elif rfm_score[2] == "5":
    return "Gros dépensiers"
elif rfm_score[0] == "1":
    return "À réactiver"
else:
    return "Clients à suivre"

rfm["segment"] = rfm["RFM_score"].apply(segmenter)

segment_colors = {
    "Champions": "#4CAF50",
    "Clients récents": "#2196F3",
    "Clients fréquents": "#FFC107",
    "Gros dépensiers": "#FF5722",
    "À réactiver": "#9C27B0",
    "Clients à suivre": "#607D8B"
}

segment_counts = rfm["segment"].value_counts().sort_values(ascending=False)
segment_percent = (segment_counts / segment_counts.sum()) * 100
colors = [segment_colors[seg] for seg in segment_counts.index]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

segment_counts.plot(kind='bar', ax=ax1, color=colors)
ax1.set_title('Nombre de clients par segment')
ax1.set_ylabel('Nombre de clients')
ax1.set_xlabel('Segment')

wedges, texts, autotexts = ax2.pie(segment_percent, labels=segment_percent.index, autopct='%1
    startangle=90, colors=colors)
```

```
centre_circle = plt.Circle((0, 0), 0.70, fc='white')  
fig.gca().add_artist(centre_circle)  
ax2.set_title('Répartition des clients (%)')  
  
plt.tight_layout()  
plt.show()
```

```
from math import pi

rfm_avg = rfm.groupby("segment")[["Recence", "Frequence", "Montant"]].mean()

for col in ["Recence", "Frequence", "Montant"]:
    rfm_avg[col] = pd.qcut(rfm_avg[col], 5, labels=[1, 2, 3, 4, 5]).astype(int)

rfm_avg["Recence"] = 6 - rfm_avg["Recence"]

categories = ["Recence", "Frequence", "Montant"]
N = len(categories)
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]

segments = rfm_avg.index.tolist()
n_segments = len(segments)
n_cols = min(3, n_segments)
n_rows = (n_segments + n_cols - 1) // n_cols

fig, axs = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows * 6),
                        subplot_kw=dict(polar=True))

if n_segments == 1:
    axs = [axs]

axs = axs.flatten()
```



```
for i, segment in enumerate(segments):
    values = rfm_avg.loc[segment].tolist()
    values += values[:1]

    ax = axs[i]
    ax.set_theta_offset(pi / 2)
    ax.set_theta_direction(-1)
    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(categories, fontsize=12)
    ax.plot(angles, values, linewidth=2, linestyle='solid', label=segment)
    ax.fill(angles, values, alpha=0.25)
    ax.set_yticks([1, 2, 3, 4, 5])
    ax.set_yticklabels(['1', '2', '3', '4', '5'], color='grey', size=10)
    ax.set_ylim(0, 5)
    ax.set_title(segment, size=16, y=1.1, pad=20)

for j in range(i + 1, len(axs)):
    axs[j].set_visible(False)

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


✓ V) Analyse de la clientèle

✓ 0) Supression Outliers(Clients par rapport au CA généré ou Nombre de vente)

```
df_merge_ss_outliers=df_merge[df_merge["type"]=="Retail"]
```

1) Analyser la distribution-confirmation normal ou non 2) Effectuer un test statistiques(parametrique ou non parametriques)

✓ 1) Corrélation Sexe & Catégorie de livre (2 variables quantitatives)

```
from scipy.stats import chi2_contingency

contingence = pd.crosstab(df_merge['sex'], df_merge['categ'])
chi2_stat, p_value, dof, expected = chi2_contingency(contingence)
```

```
sn.heatmap(contingence, annot=True, fmt='d', cmap="YlGnBu")
plt.title("Sexe vs Catégorie de livres")
plt.show()

print(f"Statistique Chi-2: {round(chi2_stat,2)}")
print(f"Valeur p: {round(p_value,4)}")
print(f"Degrés de liberté: {dof}")
print("Fréquences attendues:")
print(expected)
```

▼ 2) Corrélation age & Montant achat(2 quantitatives)

```
df_total = df_merge_ss_outliers.groupby('client_id').agg({
    'age': 'first',
    'price': 'sum'
}).reset_index()
```

Analyse de la Distribution de la variable "age"

```
from scipy.stats import shapiro
c_value,p_value=shapiro(df_total["age"])
print(f"Coefficient corrélation :{c_value}")
print(f"p-value : {p_value}")

if p_value > 0.05:
    print("La distribution semble normale (on ne rejette pas H0).")
else:
    print("La distribution n'est pas normale (on rejette H0).")
```

```
Coefficient corrélation :0.9685728710268329
```

```
p-value : 2.7868186329145244e-39
```

```
La distribution n'est pas normale (on rejette H0).
```

```
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:586: UserWarning: scipy.stats.shapiro: For N
    res = hypotest_fun_out(*samples, **kwds)
```

Test de Spearman

```
from scipy.stats import spearmanr

age=df_total["age"]
montant=df_total["price"]

s_corr,s_p_value=spearmanr(age,montant)
sn.regplot(data=df_total,x="age",y="price",robust=True, line_kws=dict(color="r"))
print(f"Coefficient de corrélation:{round(s_corr,5)}")
print(f"p_value:{round(s_p_value,5)}")
```

▼ 3) Corrélation: Age & Frequence d'achat(2 quantitatives)

```
df_frequence = df_merge_ss_outliers.groupby('client_id')['session_id'].nunique().reset_index(name='frequence')
df_frequence = df_frequence.merge(df_merge_ss_outliers[['client_id', 'age']].drop_duplicates(),
```

```
frequence=df_frequence["frequence"]
age=df_frequence["age"]
c_value,p_value=spearmanr(frequence,age)
```

```
sn.regplot(data=df_frequence,x="frequence",y="age",robust=True, line_kws=dict(color="r"))
```

```
print(f"Coefficient de corrélation:{c_value}")
print(f"p_value : {round(p_value,10)}")
```

▼ 4) Corrélation Age ↔ Taille du panier moyen (2 quantitatives)

```
panier = df_merge.groupby('client_id').agg({
    'age': 'first',
    'price': 'sum',
    'session_id': 'nunique'
})

panier['panier_moyen'] = panier['price'] / panier['session_id']

c, p = spearmanr(panier['age'], panier['panier_moyen'])

print(f"Coefficient de corrélation:{round(c,2)}")
print(f"Value-p :{p}")

sn.regplot(data=panier, x='age', y='panier_moyen', robust=True, line_kws=dict(color="r"))

plt.title(f"Corrélation âge / panier moyen (c={c:.2f}, p={p:.4f})")
plt.show()
```



```
... Coefficient de corrélation:-0.7  
Value-p :0.0
```

▼ 5) Corrélation Âge ↔ Catégorie de livres (1quali | 1 quanti)

Test de corrélation(Krussal-Wallis-> Age ne suit pas une distribution normal)

```
from scipy.stats import kruskal
```

```
cat0 = df_merge_ss_outliers[df_merge_ss_outliers['categ'] == 0]['age']
cat1 = df_merge_ss_outliers[df_merge_ss_outliers['categ'] == 1]['age']
cat2 = df_merge_ss_outliers[df_merge_ss_outliers['categ'] == 2]['age']

H,p_value=kruskal(cat0,cat1,cat2)
n=df_merge_ss_outliers.shape[0]
k=3
print(f'Coefficient de corrélation: {H}')
print(f"p-value : {p_value}")

eta_squared =H/(n-1)

print(f"  $\eta^2$ : {eta_squared}")
```

...

```
sn.boxplot(data=df_merge_ss_outliers, x='categ', y='age')
plt.title("Répartition de l'âge selon la catégorie de produit")
plt.xlabel("Catégorie de produit")
plt.ylabel("Âge des clients")
plt.show()
```

...

