Web & NetWork Data Science:

Rapport du projet:
L'application des règles d'association
&
des algorithmes de clustering

Réalisé par :

Ghazouani Farah

1 MP DSB

2024-2025

Introduction

Dans le cadre de notre projet de Web & Network Data Science, nous avons exploré un jeu de données issu d'un site e-commerce afin d'y appliquer des techniques de data cleaning, d'exploration, de clustering et d'extraction de règles d'association. Notre objectif était d'analyser le comportement des utilisateurs en ligne, d'identifier des tendances de consommation et de proposer des pistes d'optimisation basées sur les données.

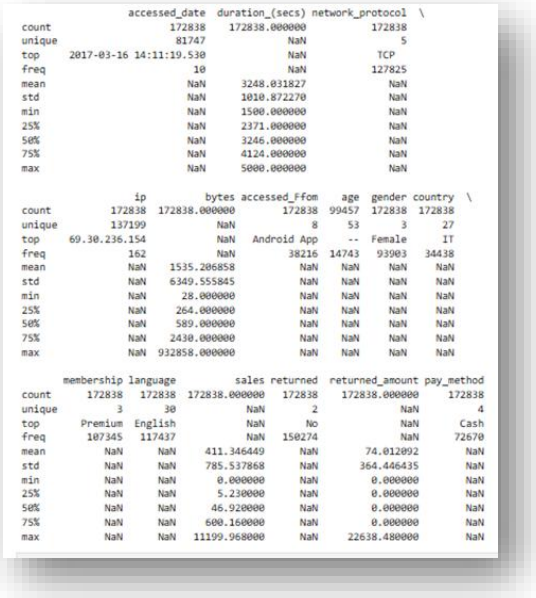
1.Choix du jeu de données

Nous avons choisi un jeu de données issu d'un site e-commerce, contenant des **logs de navigation et de transactions**. Chaque ligne correspond à une session utilisateur, avec des informations **techniques** (date d'accès, durée, protocole, navigateur, IP), **démographiques** (âge, sexe, pays, langue, abonnement) et **transactionnelles** (ventes, retours, montant échangé, méthode de paiement). Cela nous permet d'analyser le **comportement des utilisateurs**, d'évaluer les **performances commerciales** et d'identifier des **tendances** utiles pour des tâches

de **classification, clustering** ou **prédiction**.

2.Processus du traitement

Nous avons commencé par l'importation des bibliothèques nécessaires à la manipulation des données, à la visualisation, au traitement des valeurs manquantes, à la classification, à la normalisation et au clustering. Ensuite, nous avons chargé le fichier de données nommé **"E-commerce Website Logs.csv"** à l'aide de la bibliothèque **pandas**, puis affiché un **aperçu général du dataset**.

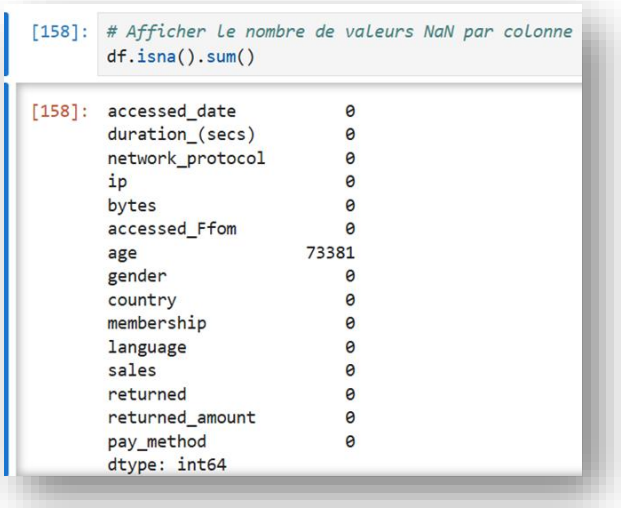


	accessed_date	duration_(secs)	network_protocol	\
count	172838	172838.000000	172838	
unique	81747	NaN	5	
top	2017-03-16 14:11:19.530	NaN	TCP	
freq	10	NaN	127825	
mean	NaN	3248.031827	NaN	
std	NaN	1010.872270	NaN	
min	NaN	1500.000000	NaN	
25%	NaN	2371.000000	NaN	
50%	NaN	3246.000000	NaN	
75%	NaN	4124.000000	NaN	
max	NaN	5000.000000	NaN	

	ip	bytes	accessed_Ffom	age	gender	country	\
count	172838	172838.000000	172838	99457	172838	172838	
unique	137199	NaN	8	53	3	27	
top	69.30.236.154	NaN	Android App	--	Female	IT	
freq	162	NaN	38216	14743	93903	34438	
mean	NaN	1535.206858	NaN	NaN	NaN	NaN	
std	NaN	6349.555845	NaN	NaN	NaN	NaN	
min	NaN	28.000000	NaN	NaN	NaN	NaN	
25%	NaN	264.000000	NaN	NaN	NaN	NaN	
50%	NaN	589.000000	NaN	NaN	NaN	NaN	
75%	NaN	2430.000000	NaN	NaN	NaN	NaN	
max	NaN	932858.000000	NaN	NaN	NaN	NaN	

	membership	language	sales	returned	returned_amount	pay_method
count	172838	172838	172838.000000	172838	172838.000000	172838
unique	3	30	NaN	2	NaN	4
top	Premium	English	NaN	No	NaN	Cash
freq	107345	117437	NaN	150274	NaN	72670
mean	NaN	NaN	411.346449	NaN	74.012092	NaN
std	NaN	NaN	785.537868	NaN	364.446435	NaN
min	NaN	NaN	0.000000	NaN	0.000000	NaN
25%	NaN	NaN	5.230000	NaN	0.000000	NaN
50%	NaN	NaN	46.920000	NaN	0.000000	NaN
75%	NaN	NaN	600.160000	NaN	0.000000	NaN
max	NaN	NaN	11199.968000	NaN	22638.480000	NaN

Nous avons constaté la présence de valeurs manquantes dans notre dataset. Pour les identifier, nous avons utilisé la fonction isna() de la bibliothèque **pandas**.



```
[158]: # Afficher le nombre de valeurs NaN par colonne
       df.isna().sum()
```

```
[158]: accessed_date           0
       duration_(secs)         0
       network_protocol        0
       ip                      0
       bytes                   0
       accessed_Ffom           0
       age                 73381
       gender                  0
       country                 0
       membership              0
       language                0
       sales                   0
       returned                0
       returned_amount         0
       pay_method              0
       dtype: int64
```

Cette figure montre qu'il y a **73 381 valeurs manquantes** dans la colonne *age*. Pour vérifier cette information, nous avons utilisé la fonction isnull().sum() afin de calculer le **nombre total de valeurs manquantes par colonne**, ainsi que leur **pourcentage par rapport au nombre total de lignes** du dataset. Cela suggère les mêmes résultats.

Tableau des valeurs manquantes:

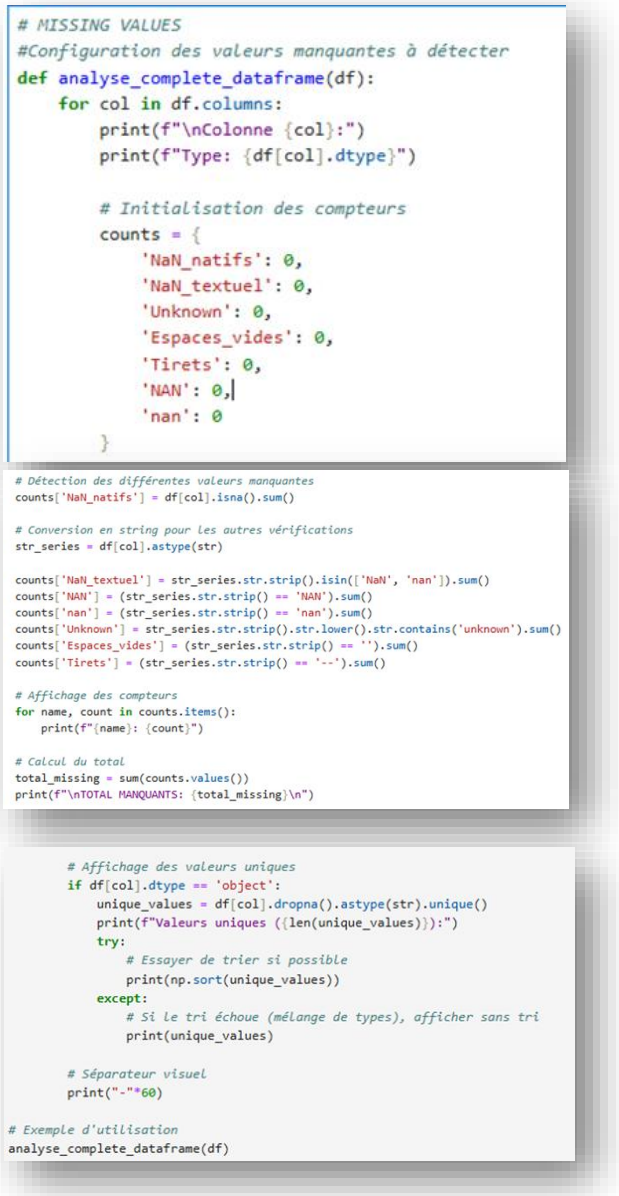
	Colonne	Valeurs manquantes	Pourcentage (%)
6	age	73381	42.46
0	accessed_date	0	0.00
1	duration_(secs)	0	0.00
2	network_protocol	0	0.00
3	ip	0	0.00
4	bytes	0	0.00
5	accessed_Ffom	0	0.00
7	gender	0	0.00
8	country	0	0.00
9	membership	0	0.00
10	language	0	0.00
11	sales	0	0.00
12	returned	0	0.00
13	returned_amount	0	0.00
14	pay_method	0	0.00

Nous avons effectué une **analyse complète des valeurs manquantes** dans chaque colonne du DataFrame, en identifiant aussi bien les formes **explicites** que **cachées**, au-delà des seuls NaN traditionnels reconnus automatiquement par pandas.

La stratégie adoptée consistait à **extraire les valeurs uniques** de chaque colonne afin de repérer les différentes représentations possibles des données manquantes. Nous avons ainsi détecté plusieurs formes :

- NaN natifs (détectés automatiquement par pandas)
- Chaînes "NaN", "nan", "NAN"
- "unknown" (insensible à la casse)
- Espaces vides ("")
- Tirets ("--")

Pour chaque colonne, nous avons affiché son **nom** et son **type** (dtype) et lancé un **compteur** des différentes formes de valeurs manquantes. Afin de garantir une détection uniforme, même pour les colonnes numériques ou mixtes, nous avons temporairement converti chaque colonne en **chaîne de caractères** lors de l'analyse.



```
# MISSING VALUES
#Configuration des valeurs manquantes à détecter
def analyse_complete_dataframe(df):
    for col in df.columns:
        print(f"\nColonne {col}:")
        print(f"Type: {df[col].dtype}")

        # Initialisation des compteurs
        counts = {
            'NaN_natifs': 0,
            'NaN_textuel': 0,
            'Unknown': 0,
            'Espaces_vides': 0,
            'Tirets': 0,
            'NAN': 0,
            'nan': 0
        }

        # Détection des différentes valeurs manquantes
        counts['NaN_natifs'] = df[col].isna().sum()

        # Conversion en string pour les autres vérifications
        str_series = df[col].astype(str)

        counts['NaN_textuel'] = str_series.str.strip().isin(['NaN', 'nan']).sum()
        counts['NAN'] = (str_series.str.strip() == 'NAN').sum()
        counts['nan'] = (str_series.str.strip() == 'nan').sum()
        counts['Unknown'] = str_series.str.strip().str.lower().str.contains('unknown').sum()
        counts['Espaces_vides'] = (str_series.str.strip() == '').sum()
        counts['Tirets'] = (str_series.str.strip() == '--').sum()

        # Affichage des compteurs
        for name, count in counts.items():
            print(f"{name}: {count}")

        # Calcul du total
        total_missing = sum(counts.values())
        print(f"\nTOTAL MANQUANTS: {total_missing}\n")

        # Affichage des valeurs uniques
        if df[col].dtype == 'object':
            unique_values = df[col].dropna().astype(str).unique()
            print(f"Valeurs uniques ({len(unique_values)}):")
            try:
                # Essayer de trier si possible
                print(np.sort(unique_values))
            except:
                # Si le tri échoue (mélange de types), afficher sans tri
                print(unique_values)

        # Séparateur visuel
        print("-"*60)

# Exemple d'utilisation
analyse_complete_dataframe(df)
```

L’analyse révèle un problème important dans la colonne *age*, qui contient 73 381 valeurs manquantes de type NaN ainsi que 14 743 entrées sous la forme "--", représentant au total 88 124 valeurs manquantes. La colonne *gender* présente quant à elle 15 886 valeurs "Unknown" (9,2 %), nécessitant également un traitement. Les autres colonnes sont globalement complètes et bien structurées.

```
Colonne gender:
Type: object
NaN_natifs: 0
NaN_textuel: 0
Unknown: 15886
Espaces_vides: 0
Tirets: 0
NAN: 0
nan: 0

TOTAL MANQUANTS: 15886

Valeurs uniques (3):
['Female' 'Male' 'Unknown']
```

Pour préparer le jeu de données à la modélisation, il est nécessaire de corriger les formats erronés dans *age*, d'imputer les valeurs manquantes, d'encoder proprement les catégories comme *gender* et de simplifier voire supprimer certaines colonnes trop spécifiques comme *ip*, qui contient 137 199 valeurs uniques et pourrait introduire du bruit dans les modèles.

3.Les méthodes utilisées pour la préparation du jeu de données

1. Imputation de la colonne Age

Pour traiter les nombreuses valeurs manquantes dans la colonne *age*, une méthode d'imputation avancée appelée **Hot Deck** a été mise en place. Le processus commence par un nettoyage approfondi de la colonne, en remplaçant toutes les valeurs aberrantes ou textuelles telles que "--", "nan", "NaN" ou encore les chaînes vides par des NaN reconnus par pandas.

```python
def clean_age_data(df, target_col='age'):
    """Nettoyage spécifique de la colonne age"""
    df = df.copy()

    # Conversion des valeurs problématiques en NaN
    age_missing_values = ['--', 'nan', 'NaN', 'N/A', 'n/a', 'NULL', 'null', '']
    df[target_col] = df[target_col].replace(age_missing_values, np.nan)

    # Conversion en numérique
    df[target_col] = pd.to_numeric(df[target_col], errors='coerce')

    return df
```

Ensuite, la méthode **Hot Deck** recherche, pour chaque ligne avec un âge manquant, les profils les plus similaires selon des critères comme le genre, le pays ou le type d'abonnement. Ces profils, appelés "donneurs", permettent d'estimer un âge réaliste à travers la médiane des âges observés chez les voisins les plus proches.

```python
def robust_hot_deck(df, target_col='age', match_cols=None, n_neighbors=5, sample_size=10000):
    """
    Version améliorée avec prétraitement de la colonne age
    """
    # 1. Nettoyage initial des données
    df = clean_age_data(df, target_col)

    if match_cols is None:
        match_cols = ['gender', 'country','membership','language', 'pay_method']


    # Vérification des colonnes
    missing_cols = [col for col in [target_col] + match_cols if col not in df.columns]
    if missing_cols:
        raise ValueError(f"Colonnes manquantes: {missing_cols}")

    # 2. Échantillonnage pour les tests
    if sample_size and len(df) > sample_size:
        df = df.sample(sample_size, random_state=42).copy()

    # 3. Séparation donneurs/receveurs
    donors = df.dropna(subset=[target_col]).copy()
    receivers = df[df[target_col].isnull()].copy()
```

Cette approche est robuste, car elle préserve les relations entre variables et maintient une cohérence statistique dans la distribution finale. Des visualisations ont permis de vérifier que la distribution des âges imputés reste fidèle à la structure initiale, ce qui est crucial pour garantir la qualité des analyses ultérieures notamment en marketing ou segmentation client.

2. Imputation de la colonne gender

Pour compléter les valeurs manquantes dans la colonne *gender*, représentées par "Unknown", une approche basée sur un **classifieur KNN** a été appliquée.

```
def impute_gender(df, target_col='gender', match_cols=None, n_neighbors=5):
    """
```

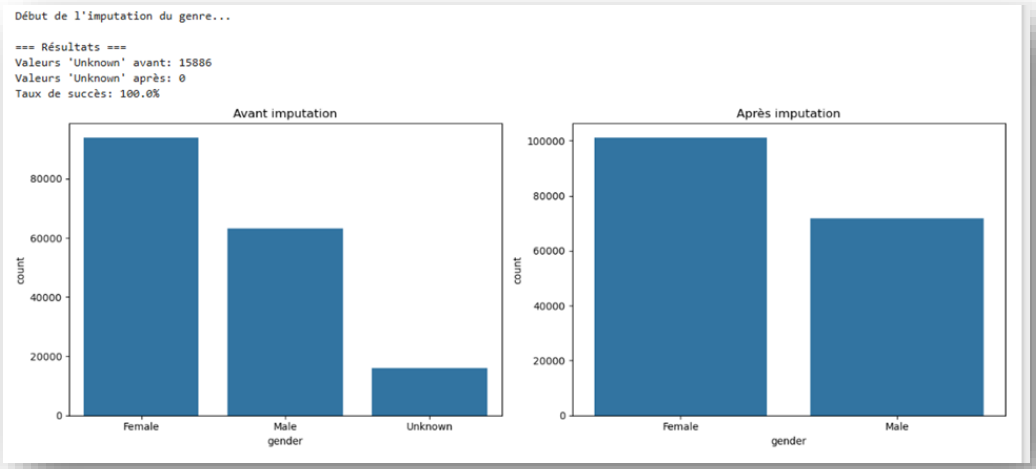
Après un nettoyage robuste des colonnes de correspondance (age, country, membership, etc.), les données sont encodées via un **OneHotEncoder** (ou **OrdinalEncoder** en cas de souci) pour permettre l'entraînement du modèle.

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(handle_unknown='ignore')
try:
    donors_encoded = encoder.fit_transform(donors[match_cols])
    receivers_encoded = encoder.transform(receivers[match_cols])
except:
    from sklearn.preprocessing import OrdinalEncoder
    encoder = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
    encoder.fit(df[match_cols])
    donors_encoded = encoder.transform(donors[match_cols])
    receivers_encoded = encoder.transform(receivers[match_cols])
```

Les lignes ayant un genre connu servent de **donneurs**, tandis que celles avec "Unknown" sont les **receveurs**. Le modèle KNN est ensuite utilisé pour prédire le genre des receveurs en se basant sur leurs caractéristiques similaires à celles des donneurs.

```
# 5. Imputation KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=min(n_neighbors, len(donors)))
knn.fit(donors_encoded, donors[target_col])
```

Cette méthode permet d'imputer intelligemment les genres manquants tout en **préservant la structure naturelle des données**. Voici une visualisation comparative avant/après montre que l'imputation réduit efficacement le nombre de "Unknown" tout en respectant la distribution initiale.



Après l'imputation des valeurs manquantes, une **exploration globale du dataset** est réalisée pour vérifier l'intégrité des données et s'assurer de la cohérence post-traitement. À l'aide de statistiques générales, on évalue le nombre total de visiteurs et la structure du jeu de données. Cette étape permet de valider que toutes les colonnes attendues sont bien présentes, que les types de données sont corrects et que les imputations précédentes ont été correctement appliquées. Elle constitue une **étape clé de validation** avant toute modélisation ou analyse plus poussée.

Cette figure représente les statistiques globales :

```
📊 STATISTIQUES GLOBALES
Nombre total de visiteurs: 172838

Structure des données:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 172838 entries, 0 to 172837
Data columns (total 15 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   accessed_date     172838 non-null  object
 1   duration_(secs)   172838 non-null  int64
 2   network_protocol  172838 non-null  object
 3   ip                172838 non-null  object
 4   bytes             172838 non-null  int64
 5   accessed_Ffom     172838 non-null  object
 6   age               172838 non-null  float64
 7   gender            172838 non-null  object
 8   country           172838 non-null  object
 9   membership        172838 non-null  object
 10  language          172838 non-null  object
 11  sales             172838 non-null  float64
 12  returned          172838 non-null  object
 13  returned_amount   172838 non-null  float64
 14  pay_method        172838 non-null  object
dtypes: float64(3), int64(2), object(10)
memory usage: 19.8+ MB
None
```

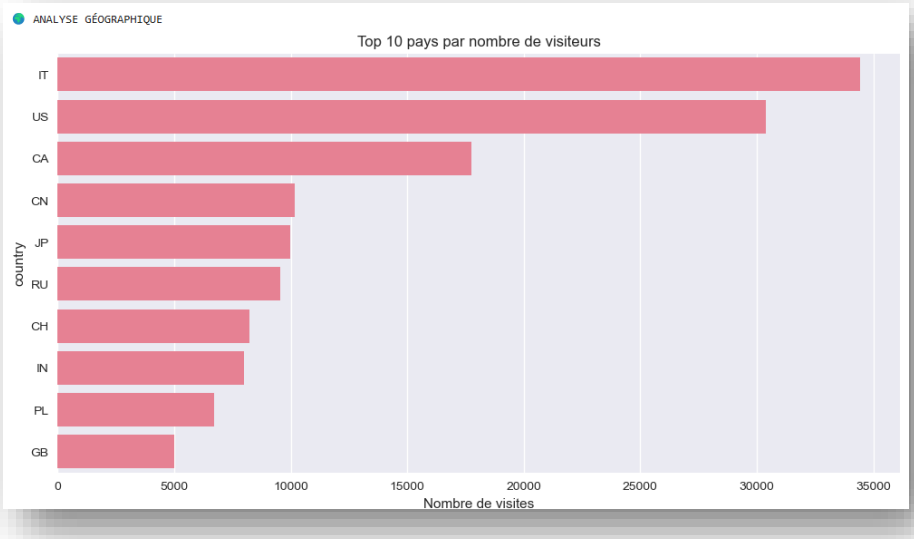
4.Exploration de données

- **Top 10 pays par nombre de visiteurs**

Nous avons suggéré une visualisation pour représenter les **10 pays les plus présents** dans notre dataset en termes de nombre de visiteurs à l'aide d'un graphique en barres horizontales.

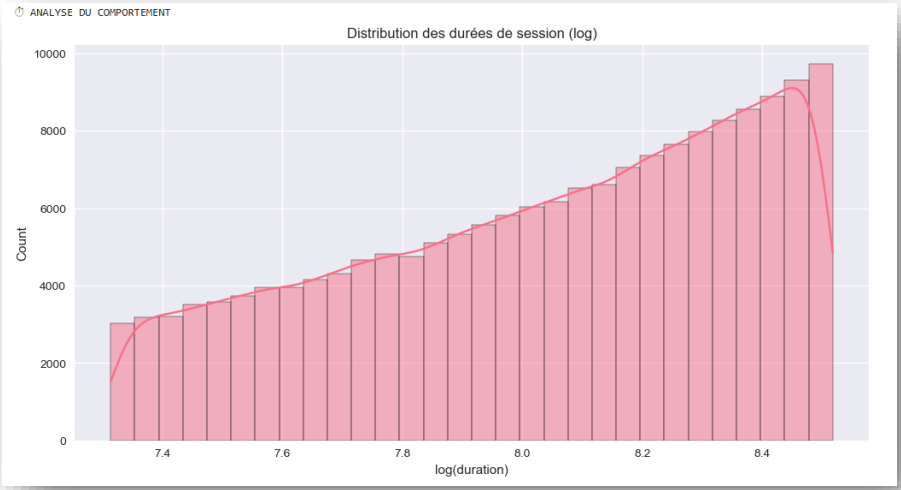
```
# Top 10 pays
top_countries = df_imputed['country'].value_counts().nlargest(10)
plt.figure(figsize=(12, 6))
sns.barplot(x=top_countries.values, y=top_countries.index, orient='h')
plt.title('Top 10 pays par nombre de visiteurs')
plt.xlabel('Nombre de visites')
plt.show()
```

Les pays les plus représentés sont l'Italie 'IT', États-Unis 'US', Canada 'CA', Chine 'CN' et Japon 'JP'. Cela suggère des **opportunités marketing ciblées** dans ces zones. L'absence de pays clés comme **France** ou **Allemagne** peut indiquer un **biais géographique** ou un **problème de tracking**, à explorer.



- **Distribution des durées de session (log)**

Nous avons également réalisé une analyse des durées de session pour montrer que la majorité des utilisateurs ayant des durées de session relativement fortes. L'échelle logarithmique permet de mieux visualiser les longues sessions.



- **Répartition des méthodes de paiement**

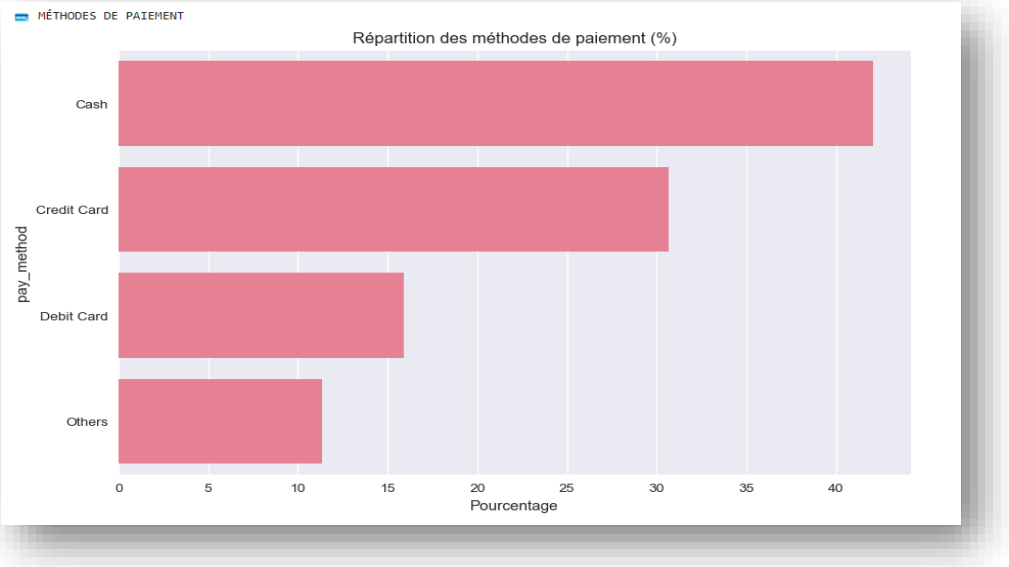
Nous avons obtenu un taux de conversion élevé à **76.08%**, indiquant qu'une grande majorité des utilisateurs effectuent un achat après avoir visité la plateforme. Ce résultat suggère que l'expérience utilisateur est efficace et que les stratégies de conversion mises en place sont performantes.

```
# 5. Analyse des ventes
# ============================================
print("\n💰 ANALYSE DES VENTES")

# Taux de conversion
conversion_rate = (df_imputed['sales'] > 0).mean() * 100
print(f"Taux de conversion: {conversion_rate:.2f}%")
```

```
💰 ANALYSE DES VENTES
Taux de conversion: 76.08%
```

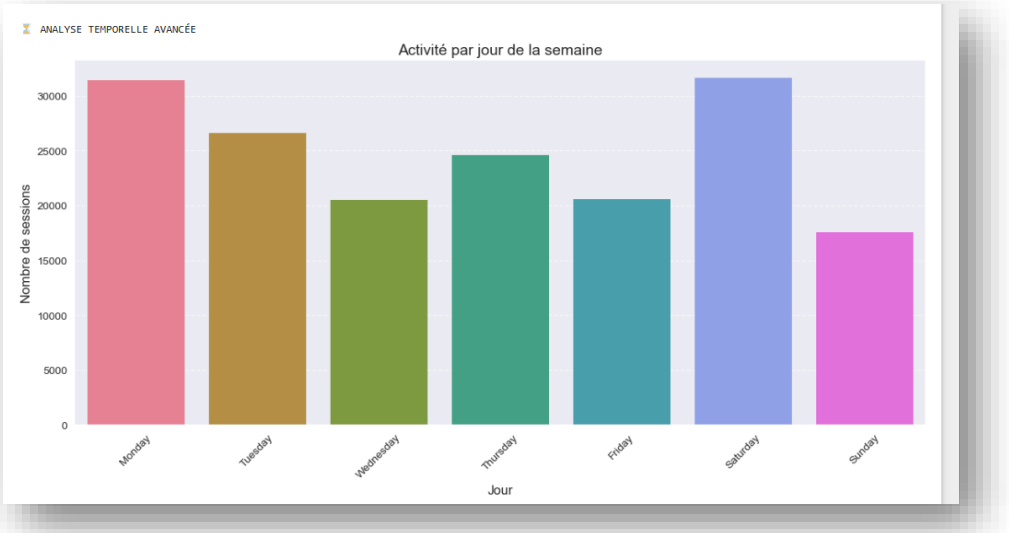
En termes de **méthodes de paiement**, la majorité des utilisateurs (environ **45%**) privilégient le **cash**, suivi de près par **32%** des utilisateurs optant pour la **carte de crédit**. Cependant, seulement **17%** des utilisateurs utilisent la **carte de débit**. Ces résultats suggèrent que les paiements en espèces et par carte de crédit dominent, tandis que l'usage de la carte de débit reste relativement

faible. Il serait pertinent d'examiner les raisons derrière cette préférence pour le cash et la carte de crédit, et de cibler des actions marketing pour encourager l'adoption des cartes de débit.



- **Distribution des activités par jour de la semaine**

L'analyse temporelle avancée permet d'explorer les tendances d'activité tout au long de la semaine. Il en ressort que **Saturday**, **Monday**, **Tuesday**, et **Thursday** enregistrent le plus grand nombre de sessions, mettant en évidence des pics d'engagement durant ces jours spécifiques.



5.Application des algorithmes Apriori et FP-Growth

Afin d'extraire des relations significatives entre les comportements utilisateurs, deux algorithmes d'exploration d'associations ont été appliqués : **Apriori** et **FP-Growth**. Les données ont d'abord été transformées en une structure transactionnelle en regroupant, pour chaque session (ip et accessed_date), les méthodes de paiement utilisées, ainsi que le genre, le pays et le type d'abonnement de l'utilisateur.

```
# 1. Préparer les données en format transactionnel
transactions = df_imputed.groupby(['ip', 'accessed_date'], group_keys=False).apply(
    lambda x: list(x['pay_method']) +
              [f"gender_{x['gender'].iloc[0]}"] +
              [f"country_{x['country'].iloc[0]}"] +
              [f"membership_{x['membership'].iloc[0]}"]
).tolist()
```

Ces transactions ont été encodées en variables booléennes à l'aide de TransactionEncoder.

```
# Encoder les données transactionnelles
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)
```

Ensuite, l'algorithme Apriori a été utilisé pour identifier les itemsets fréquents (avec un support minimal de 5%) et en déduire des règles d'association, triées par valeur de lift et de confiance.

```
# 2. Appliquer l'algorithme Apriori
frequent_itemsets = apriori(df_encoded, min_support=0.05, use_colnames=True)

# 3. Générer les règles d'association avec Apriori
rules_apriori = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

# Trier les règles par lift et confiance décroissante
rules_apriori_sorted = rules_apriori.sort_values(by=['lift', 'confidence'], ascending=[False, False])

# Afficher les 10 règles les plus importantes
print("\n🔹 Top 10 des règles générées par l'algorithme Apriori :")
print(rules_apriori_sorted.head(10))
```

En parallèle, FP-Growth a permis une génération plus rapide des motifs fréquents et des règles associées (confiance ≥ 0.7).

```python
# 4. Appliquer FP-Growth
transactions_fpgrowth = [x for x in transactions]
min_support = 0.05  # Support minimum souhaité
patterns = pyfpgrowth.find_frequent_patterns(transactions_fpgrowth, min_support)

# 5. Générer les règles d'association avec FP-Growth
confidence_threshold = 0.7
rules_fpgrowth = pyfpgrowth.generate_association_rules(patterns, confidence_threshold)

# Trier les règles FP-Growth par niveau de confiance décroissant
rules_fpgrowth_sorted = sorted(rules_fpgrowth.items(), key=lambda x: x[1], reverse=True)
```

Les dix règles les plus pertinentes pour chaque méthode ont été affichées afin d'identifier des patterns comportementaux récurrents.

```python
# Afficher les 10 règles les plus importantes
print("\n🔹 Top 10 des règles générées par FP-Growth :")
for rule, confidence in rules_fpgrowth_sorted[:10]:
    print(f"Règle: {rule} → Confiance: {confidence}")
```

- **Interprétation des résultats**

Les résultats obtenus à l'aide des algorithmes Apriori et FP-Growth mettent en lumière des relations intéressantes entre les caractéristiques des utilisateurs et leurs comportements d'achat. Par exemple, Apriori révèle que les femmes italiennes utilisent fréquemment le paiement en espèce (36), ou encore que les femmes canadiennes sont fortement représentées parmi les utilisateurs (17). Des règles comme celle reliant les femmes américaines au statut Premium (78) indiquent des profils d'abonnement spécifiques selon des caractéristiques démographiques.

🔹 Top 10 des règles générées par l'algorithme Apriori :

	antecedents	consequents
33	(country_IT, gender_Female)	(Cash)
36	(Cash)	(country_IT, gender_Female)
57	(membership_Normal)	(Credit Card, gender_Female)
56	(Credit Card, gender_Female)	(membership_Normal)
16	(country_CA)	(gender_Female)
17	(gender_Female)	(country_CA)
32	(country_IT, Cash)	(gender_Female)
37	(gender_Female)	(country_IT, Cash)
78	(membership_Premium)	(country_US, gender_Female)
77	(country_US, gender_Female)	(membership_Premium)

Ces règles affichent une **confiance modérée à élevée** et un **lift supérieur à 1**, ce qui témoigne d'une corrélation positive entre les items.

	antecedent support	consequent support	support	confidence	lift
33	0.114050	0.430720	0.052870	0.463565	1.076258
36	0.430720	0.114050	0.052870	0.122747	1.076258
57	0.284649	0.180470	0.054261	0.190623	1.056262
56	0.180470	0.284649	0.054261	0.300664	1.056262
16	0.103071	0.585480	0.063366	0.614782	1.050048
17	0.585480	0.103071	0.063366	0.108230	1.050048
32	0.086329	0.585480	0.052870	0.612417	1.046008
37	0.585480	0.086329	0.052870	0.090301	1.046008
78	0.623758	0.100831	0.065408	0.104861	1.039969
77	0.100831	0.623758	0.065408	0.648689	1.039969

De son côté, **FP-Growth** a généré des règles très précises, avec une **confiance maximale de 1.0**, comme par exemple : les hommes français utilisant "Cash" et "Others" comme méthodes de paiement sont toujours associés au statut **Premium**. Cela suggère des segments d'utilisateurs très ciblés et homogènes.

```
🔹 Top 10 des règles générées par FP-Growth :
Règle: ('Cash', 'Others', 'country_FR', 'gender_Male') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Debit Card', 'Debit Card', 'country_DK', 'gender_Male') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Cash', 'Credit Card', 'country_AE', 'gender_Female') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Cash', 'Cash', 'country_AE', 'gender_Female') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Cash', 'Others', 'country_IE', 'gender_Female') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Cash', 'Others', 'country_IE', 'gender_Male') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Debit Card', 'Others', 'country_IE', 'gender_Male') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Cash', 'Debit Card', 'country_IE', 'gender_Female') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Debit Card', 'Debit Card', 'country_IE', 'gender_Female') → Confiance: (('membership_Premium',), 1.0)
Règle: ('Credit Card', 'Credit Card', 'country_IE', 'gender_Male') → Confiance: (('membership_Premium',), 1.0)
```

L'application des deux algorithmes permet de **valider** les tendances générales (avec Apriori), puis de **creuser plus profondément** pour découvrir des règles précises et fortement fiables (avec FP-Growth). Cela renforce la robustesse de l'analyse et permet de mieux cibler les recommandations marketing ou les décisions stratégiques en fonction des profils utilisateurs.

Afin d'**identifier les associations fréquentes entre les caractéristiques des utilisateurs et leurs comportements d'achat**, ce qui aide à mieux comprendre les profils clients et à orienter les stratégies marketing ou de personnalisation.
Nous avons préparé les données en sessions clients (par IP et date) et transformé chaque session en une transaction contenant les méthodes de paiement, le genre, le pays et le type d'abonnement.

```python
# Création des transactions (1 transaction = 1 session client)
transactions = df_imputed.groupby(['ip', 'accessed_date']).agg(
    pay_method=('pay_method', list),
    gender=('gender', 'first'),
    country=('country', 'first'),
    membership=('membership', 'first')
).apply(lambda row: row['pay_method'] +
                    [f"gender_{row['gender']}"] +
                    [f"country_{row['country']}"] +
                    [f"membership_{row['membership']}"], axis=1).tolist()
```

Ces transactions sont ensuite encodées sous forme binaire grâce à TransactionEncoder, ce qui permet d'appliquer l'algorithme **Apriori**.

```python
# Encodage
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)
```

Ensuite nous avons identifié les combinaisons d'éléments (itemsets) les plus fréquentes dans les données, servant de base pour découvrir des règles d'association représentatives des habitudes des utilisateurs.

```
# Appliquer Apriori pour extraire les itemsets fréquents
frequent_itemsets = apriori(df_encoded, min_support=0.05, use_colnames=True)

# Afficher les itemsets fréquents trouvés
print("Itemsets fréquents générés par Apriori :")
print(frequent_itemsets.head(30))
```

Nous avons alors constaté que **le moyen de paiement "Cash" est utilisé dans 43% des cas**, que **les femmes représentent 58% des utilisateurs** et que **la majorité des clients ont un abonnement "Premium" (62%)**. Certaines combinaisons apparaissent également souvent, comme **"Cash" utilisé par des femmes (25%)** ou **"Credit Card" par des femmes (18%)**, ce qui révèle des habitudes d'achat selon le genre, le pays ou le type de membre. Ces itemsets permettent d'orienter les actions ciblées et les recommandations.

```
Itemsets fréquents générés par Apriori :
     support                            itemsets
0   0.430720                             (Cash)
1   0.315562                      (Credit Card)
2   0.164812                       (Debit Card)
3   0.117772                           (Others)
4   0.103071                       (country_CA)
5   0.058771                       (country_CN)
6   0.199422                       (country_IT)
7   0.057519                       (country_JP)
8   0.055212                       (country_RU)
9   0.175519                       (country_US)
10  0.585480                    (gender_Female)
11  0.414520                      (gender_Male)
12  0.284649                (membership_Normal)
13  0.091593         (membership_Not Logged In)
14  0.623758               (membership_Premium)
15  0.086329                 (country_IT, Cash)
16  0.073972                 (country_US, Cash)
17  0.257284              (Cash, gender_Female)
18  0.173436                (gender_Male, Cash)
19  0.123186           (membership_Normal, Cash)
20  0.267769          (membership_Premium, Cash)
21  0.062686           (country_IT, Credit Card)
22  0.056146           (country_US, Credit Card)
23  0.180470        (Credit Card, gender_Female)
24  0.135092          (gender_Male, Credit Card)
25  0.090617    (membership_Normal, Credit Card)
26  0.195845   (membership_Premium, Credit Card)
27  0.096947         (gender_Female, Debit Card)
28  0.067865           (gender_Male, Debit Card)
29  0.103445    (membership_Premium, Debit Card)
```

Nous avons également appliqué l'algorithme **FP-Growth** pour identifier les **itemsets fréquents** (groupes d'attributs souvent observés ensemble) dans les transactions client. Avec un **support minimum de 2%** et une **taille maximale de 4 éléments**, cela explore efficacement les associations récurrentes dans les données encodées. L'objectif est d'obtenir rapidement les combinaisons les plus significatives sans passer par des étapes coûteuses comme dans Apriori.

```python
from mlxtend.frequent_patterns import fpgrowth

# Paramètres optimisés
min_support = 0.02  # Support minimum (2%)
max_len = 4         # Taille maximale des itemsets

frequent_itemsets = fpgrowth(
    df_encoded,
    min_support=min_support,
    use_colnames=True,
    max_len=max_len
)

print(f"Itemsets fréquents trouvés : {len(frequent_itemsets)}")
print(frequent_itemsets.sort_values('support', ascending=False).head(10))
```

Les résultats de FP-Growth montrent par exemple que la majorité des sessions proviennent de membres Premium (62%) et la majorite des clients actifs sont des femmes (58%), beaucoup paient en **Cash** (43%) et certaines combinaisons comme **"membership_Premium + gender_Female"** (37%) ou **"membership_Premium + Cash"** (27%) reviennent souvent.

```
Itemsets fréquents trouvés : 140
     support                                itemsets
3   0.623758                (membership_Premium)
4   0.585480                     (gender_Female)
5   0.430720                              (Cash)
0   0.414520                        (gender_Male)
44  0.373207  (membership_Premium, gender_Female)
10  0.315562                        (Credit Card)
1   0.284649                   (membership_Normal)
45  0.267769           (membership_Premium, Cash)
46  0.257284                (Cash, gender_Female)
24  0.250551    (membership_Premium, gender_Male)
```

Nous avons par la suite extraire des **règles d'association intéressantes** à partir des itemsets fréquents générés par FP-Growth qui utilise une **confiance minimale de 0.3**, ce qui permet d'inclure plus de règles même moins fortes pour une exploration plus large.

```
temp_rules = association_rules(
    frequent_itemsets,
    metric="confidence",
    min_threshold=0.3  # Réduire la confiance minimum
)
```

Ensuite, il affiche la **distribution des valeurs de lift et de confiance**, afin d'avoir une vue statistique sur la qualité globale des règles. Enfin, il filtre les **règles les plus pertinentes** en ne conservant que celles avec un **lift supérieur à la médiane** (ce qui est plus utiles que la moyenne) et ayant **un seul élément dans la conséquence**, pour une meilleure lisibilité et interprétation.

```python
print("Distribution du lift:", temp_rules['lift'].describe())
print("Distribution de la confiance:", temp_rules['confidence'].describe())

# Ajuster dynamiquement les seuils
median_lift = temp_rules['lift'].median()
interesting_rules = temp_rules[
    (temp_rules['lift'] > median_lift) &
    (temp_rules['consequents'].apply(len) == 1)
]
```

La distribution du **lift** montre que la majorité des règles sont proches de 1, avec une moyenne de **1.00** et un maximum de **1.30**, ce qui indique que peu de règles ont une dépendance forte entre antécédent et conséquent. Concernant la **confiance**, la moyenne est de **0.49**, ce qui reflète une fiabilité modérée des règles extraites. En fixant un seuil dynamique basé sur la médiane du lift (1.00), on sélectionne uniquement les règles **plus intéressantes que la moyenne**, c'est-à-dire celles montrant une véritable association au-delà du hasard.

```
Distribution du lift: count    182.000000
mean       1.003044
std        0.044734
min        0.785439
25%        0.984126
50%        1.000009
75%        1.020248
max        1.303052
Name: lift, dtype: float64
Distribution de la confiance: count    182.000000
mean       0.493500
std        0.115544
min        0.300084
25%        0.402045
50%        0.444950
75%        0.598750
max        0.679043
Name: confidence, dtype: float64
```

Pour identifier les relations les plus pertinentes entre les différentes variables (comme **moyens de paiement**, **genre**, **pays**, etc) nous avons formaté les règles extraites, en transformant les antécédents et les conséquences en chaînes lisibles.
Puis nous avons fait une trie les règles les plus intéressantes par leur **lift** et affiché les 10 premières règles sous forme d'une table, incluant les colonnes **support**, **confiance** et **lift**.

```python
# Formatage des résultats
def format_rule(rule):
    ante = ", ".join(list(rule['antecedents']))
    cons = list(rule['consequents'])[0]
    return f"{ante} => {cons}"

interesting_rules['rule'] = interesting_rules.apply(format_rule, axis=1)

# Top 10 des règles par lift
top_rules = interesting_rules[['rule', 'support', 'confidence', 'lift']].head(10)

print("Top 10 des règles d'association :")
print(top_rules.to_string(index=False))
```

Les résultats révèlent des liens entre des variables comme le type d'adhésion, le genre et le mode de paiement. Par exemple, les utilisateurs ayant un **membership normal** sont souvent associés à l'usage de **Cash** (confiance de 43%) avec un lift de 1.0047, indiquant une relation modérée.
Les utilisateurs avec un **membership normal** sont également plus susceptibles d'être des **femmes** (confiance de 60% et lift de 1.02), suggérant une tendance marquée parmi les membres féminins…

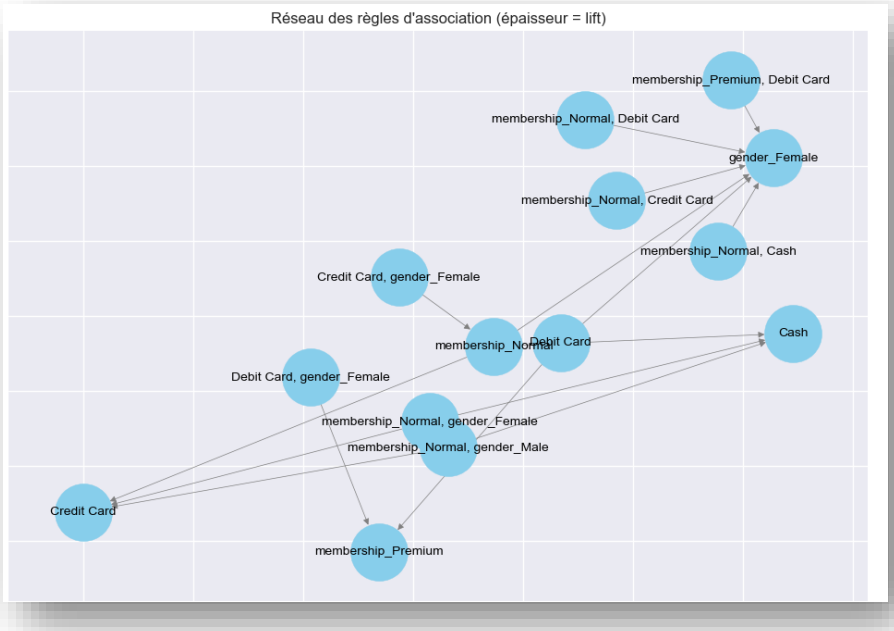
```
Top 10 des règles d'association :
                                          rule  support  confidence     lift
                     membership_Normal => Cash 0.123186    0.432764 1.004745
             membership_Normal => gender_Female 0.170154    0.597766 1.020984
               membership_Normal => Credit Card 0.090617    0.318347 1.008828
         membership_Normal, gender_Male => Cash 0.049509    0.432411 1.003927
       membership_Normal, Cash => gender_Female 0.073677    0.598093 1.021543
       membership_Normal, gender_Female => Cash 0.073677    0.433001 1.005296
membership_Normal, Credit Card => gender_Female 0.054261    0.598790 1.022734
membership_Normal, gender_Female => Credit Card 0.054261    0.318893 1.010557
Credit Card, gender_Female => membership_Normal 0.054261    0.300664 1.056262
  membership_Normal, gender_Male => Credit Card 0.036357    0.317536 1.006258
```

Nous avons ainsi crée un graphe dirigé (DiGraph) pour représenter les règles d'association, où chaque nœud représente un itemset (composant de la règle) et chaque arête (flèche) indique une relation entre l'antécédent et le conséquent d'une règle. L'épaisseur des arêtes est proportionnelle à la valeur de **lift**, ce qui permet de visualiser la force des associations (Les règles avec un lift plus élevé auront des arêtes plus épaisses, permettant de repérer facilement les relations les plus significatives dans le réseau).

```
]: import networkx as nx

   # Création du graphe
   G = nx.DiGraph()
   for _, rule in interesting_rules.head(15).iterrows():
       G.add_edge(
           ", ".join(list(rule['antecedents'])),
           list(rule['consequents'])[0],
           weight=rule['lift']
       )

   # Dessin du graphe
   plt.figure(figsize=(12, 8))
   pos = nx.spring_layout(G, k=0.5)
   nx.draw_networkx(
       G, pos,
       with_labels=True,
       node_size=2000,
       node_color='skyblue',
       font_size=10,
       edge_color='gray',
       width=[d['weight']*0.5 for (u,v,d) in G.edges(data=True)]
   )
   plt.title("Réseau des règles d'association (épaisseur = lift)")
   plt.show()
```

Chaque cercle bleu symbolise un élément (comme un mode de paiement, un type d'abonnement ou le genre d'un client), tandis que les flèches indiquent des liens de probabilité entre eux, révélés par une analyse de règles d'association.

Par exemple, on observe que l'utilisation de "Cash" est fortement associée à un abonnement "Normal" et que les personnes ayant un abonnement "Premium" sont plus susceptibles d'utiliser une carte de crédit ou de débit.



6.Application du clustering

Nous avons commencé par convertir une variable catégorielle "gender" en valeurs numériques (0 pour "Female" et 1 pour "Male"). Ensuite, les données sont standardisées pour que chaque variable ait une moyenne de 0 et un écart-type de 1 ce qui est essentiel pour les algorithmes de clustering comme K-means et DBSCAN.

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn.decomposition import PCA
from sklearn.neighbors import NearestNeighbors

# 1. Préparation des données
features = ['duration_(secs)', 'sales', 'returned_amount', 'age', 'gender']

# Conversion de la variable catégorielle 'gender' en numérique
df_imputed['gender_numeric'] = df_imputed['gender'].map({'Female': 0, 'Male': 1, 'F': 0, 'M': 1})

# Mise à jour des features avec la version numérique
features_numeric = ['duration_(secs)', 'sales', 'returned_amount', 'age', 'gender_numeric']

# Standardisation
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_imputed[features_numeric])
```

Pour l'application de k-means, une méthode du coude a été utilisée pour déterminer le nombre optimal de clusters (k) en calculant la somme des carrés des erreurs internes (WCSS) pour différentes valeurs de k (de 1 à 10).
Après avoir déterminé que k=3 est optimal, le modèle K-means est appliqué pour affecter chaque point de données à un cluster.

```python
# 2. K-means - détermination du nombre optimal de clusters
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Méthode du Coude')
plt.xlabel('Nombre de clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()

# Application avec k=3
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_imputed['cluster_kmeans'] = kmeans.fit_predict(X_scaled)
```

Un autre algorithme de clustering a été utilisé "DBSCAN" car il est capable de détecter des points atypiques (outliers).
Cela identifie les points atypiques (qui sont affectés à un cluster avec la valeur -1) et les affiche.

```python
# 3. DBSCAN - détermination des paramètres
neigh = NearestNeighbors(n_neighbors=5)
nbrs = neigh.fit(X_scaled)
distances, _ = nbrs.kneighbors(X_scaled)
distances = np.sort(distances[:, -1], axis=0)


# Paramètres DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
df_imputed['cluster_dbscan'] = dbscan.fit_predict(X_scaled)
```

Ensuite, nous avons affiché les profils des clusters générés par K-means, en calculant la moyenne des caractéristiques pour chaque cluster et en affichant le nombre de points atypiques détectés par DBSCAN.

```python
# 4. Analyse des résultats
print("Profil des clusters (K-means):")
print(df_imputed.groupby('cluster_kmeans')[features_numeric].mean())

print("\nPoints atypiques (DBSCAN):")
print(f"Nombre: {(df_imputed['cluster_dbscan'] == -1).sum()}")
```

Une réduction de dimension est effectuée à l'aide de PCA (Analyse en Composantes Principales) pour visualiser les clusters dans un espace bidimensionnel.Cela nous permet de produire deux visualisations une pour les clusters K-means et une pour les clusters DBSCAN.

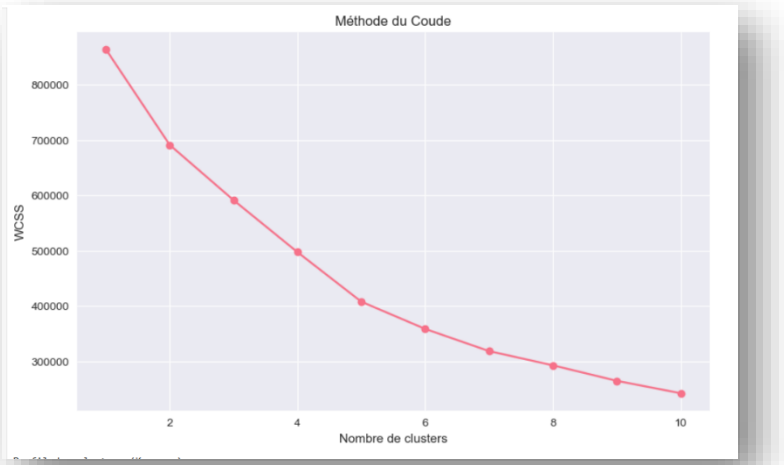
```python
# 5. Visualisation avec PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(16, 6))

plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_imputed['cluster_kmeans'], cmap='viridis', alpha=0.6)
plt.title('K-means (k=3)')

plt.subplot(1, 2, 2)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_imputed['cluster_dbscan'], cmap='viridis', alpha=0.6)
plt.title('DBSCAN')

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

Enfin, les résultats sont exportés dans un fichier CSV avec les clusters associés à chaque point de données.

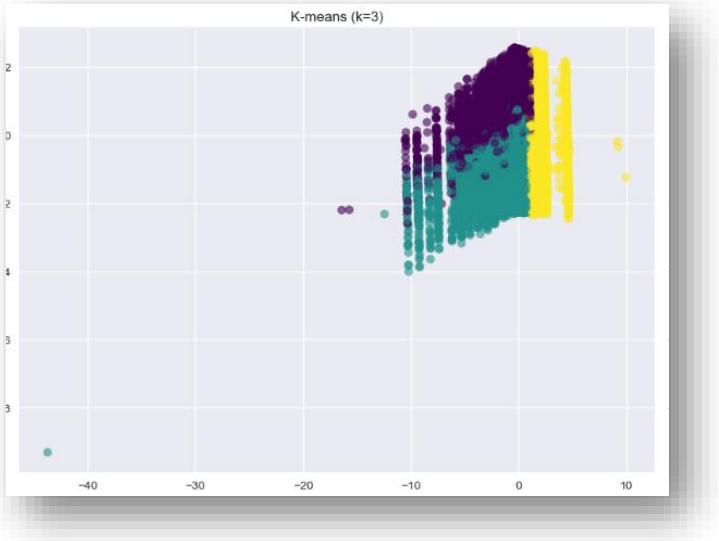
```python
# 6. Export
df_imputed.to_csv('clusters_resultsss.csv', index=False)
```

- **Interprétation des résultats**

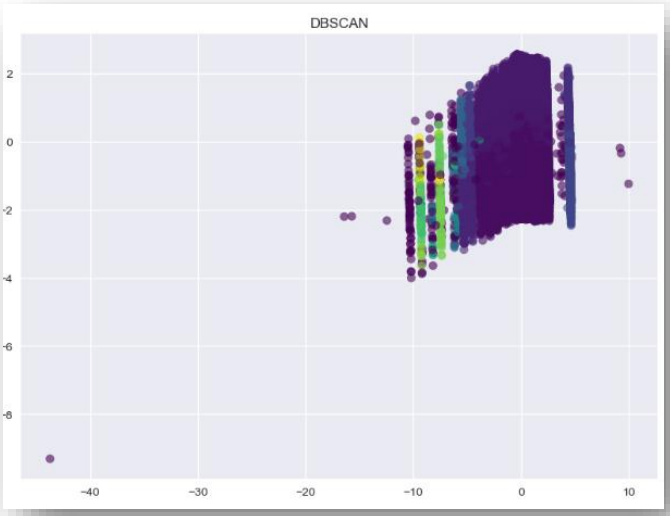
Nous avons détecté une diminution rapide du WCSS jusqu'à environ k=3, après que la courbe commence à ralentir. Cela indique que 3 clusters est un bon choix pour segmenter les données.



Une autre figure correspond à l'algorithme k-means confirme le choix de trois clusters distincts avec des couleurs différentes, (ce qui correspond à notre choix de k=3).



L’algorithme DBSCAN détecte aussi des groupes mais gère mieux les points atypiques.



Alors on distingue 3 clusters
Cluster 0 : Majoritairement des hommes, avec des ventes modérées et un montant de retour moyen.
Cluster 1 : Majoritairement des femmes, avec des ventes plus faibles et un montant de retour élevé.
Cluster 2 : Groupe mixte avec des ventes très élevées et aucun retour.

```
Profil des clusters (K-means):
                duration_(secs)        sales  returned_amount        age  \
cluster_kmeans
0                   3247.683921   238.834490        75.806194  42.846546
1                   3248.860019   168.936142        87.227776  43.051779
2                   3245.167126  2306.739717         0.000000  43.213490

                gender_numeric
cluster_kmeans
0                      1.00000
1                      0.00000
2                      0.26284
```

Conclusion

Ce projet nous a permis de mettre en pratique différentes techniques de data science, allant du traitement des données manquantes à l'application d'algorithmes d'association et de clustering. Les résultats obtenus ont révélé des patterns clients utiles pour une meilleure segmentation et des

actions marketing ciblées. Ce travail renforce notre compréhension de l'importance de la qualité des données et de l'analyse comportementale en contexte e-commerce.