Projet: Détection de Fraude sur une Plateforme E-commerce

Nom : ETSE Kossivi
Formation : Master 2 Mathématiques Appliquées, Statistiques, Parcours Data Science
Université : Aix-Marseille Université, Faculté des Sciences, Site de Saint-Charles
Année Universitaire : 2024–2025
Date : 17 Décembre 2024

I. Introduction

La détection des transactions frauduleuses constitue un enjeu majeur pour les plateformes d'e-commerce, confrontées à un volume croissant de données et à des techniques de fraude de plus en plus sophistiquées. L'identification proactive des comportements frauduleux est essentielle pour limiter les pertes financières et renforcer la confiance des utilisateurs.

Le dataset fourni contient des informations variées sur les transactions, incluant des caractéristiques liées aux utilisateurs, aux paiements et aux comportements, ce qui permettra d'identifier les signaux potentiels de fraude. L'objectif est de développer un modèle prédictif robuste et performant, capable de différencier les transactions légitimes des transactions frauduleuses.

1. Importation des bibliothèques nécessaires

In [36]:
```
# Manipulation des données
import numpy as np
import pandas as pd

# Visualisation des données
import matplotlib.pyplot as plt
import seaborn as sns

# Prétraitement et transformation des données
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

# Modélisation
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier, GradientBoosti

# Évaluation des modèles
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, classification_report, confusion_matri
)

# Tests statistiques
from scipy.stats import f_oneway, chi2_contingency

# Outils supplémentaires
import warnings
warnings.filterwarnings("ignore")

# Paramètres d'affichage pour les visualisations
sns.set_style("whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)

# Vérification
print("Tous les packages nécessaires ont été importés.")
```

```
Tous les packages nécessaires ont été importés.
```

2. Importation du jeux de données

In [37]:

```
def load_dataset(train_path, test_path):
    """
    Chargement des jeux de données d'entraînement et de test à part

    Parameters:
        train_path (str): Chemin vers le fichier CSV contenant les
        test_path (str): Chemin vers le fichier CSV contenant les d

    Returns:
        tuple: Un tuple contenant deux DataFrames (train_data, test
    """
    try:
        train_data = pd.read_csv(train_path)
        test_data = pd.read_csv(test_path)
        print("Jeux de données chargés avec succès.")

        # Affichage des informations générales sur les datasets
        print("\nInformations sur les données d'entraînement :")
        print(train_data.info())  # Affiche les informations sur tr
        print("\nInformations sur les données de test :")
        print(test_data.info())  # Affiche les informations sur tes

        return train_data, test_data
    except FileNotFoundError as e:
        print(f"Erreur : fichier introuvable. {e}")
        return None, None
    except Exception as e:
        print(f"Erreur inattendue lors du chargement des données :
        return None, None
```

```python
# Chemins vers les fichiers de données
train_file = "Train.csv"
test_file = "Test.csv"

# Chargement des données
train_data, test_data = load_dataset(train_file, test_file)

# Affichage des premières lignes pour vérifier le chargement
if train_data is not None and test_data is not None:
    print("\nAperçu des données d'entraînement :")
    print(train_data.head())
    print("\nAperçu des données de test :")
    print(test_data.head())
```

```
Jeux de données chargés avec succès.

Informations sur les données d'entraînement :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24000 entries, 0 to 23999
Data columns (total 19 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   transaction_id                   24000 non-null  int64
 1   cart_size                        24000 non-null  int64
 2   transaction_amount               24000 non-null  int64
 3   transaction_type                 24000 non-null  object
 4   transaction_status               24000 non-null  object
 5   customer_age_group               8727 non-null   object
 6   session_duration                 22665 non-null  float64
 7   time_spent_on_payment_page       15841 non-null  float64
 8   customer_ip_location             24000 non-null  object
 9   card_expiration_delay            24000 non-null  int64
 10  payment_method                   24000 non-null  object
 11  login_status                     24000 non-null  object
 12  time_since_account_creation      24000 non-null  int64
 13  ip_address_previous_transactions 24000 non-null  int64
 14  user_segment                     11384 non-null  object
 15  product_views_during_session     22665 non-null  float64
 16  visit_origin                     22665 non-null  object
 17  device_type                      22665 non-null  object
 18  flag                             24000 non-null  int64
dtypes: float64(3), int64(7), object(9)
memory usage: 3.5+ MB
None

Informations sur les données de test :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 18 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   transaction_id                   6000 non-null   int64
 1   cart_size                        6000 non-null   int64
 2   transaction_amount               6000 non-null   int64
 3   transaction_type                 6000 non-null   object
```

```
 4   transaction_status                6000 non-null   object
 5   customer_age_group                2169 non-null   object
 6   session_duration                  5689 non-null   float64
 7   time_spent_on_payment_page        3997 non-null   float64
 8   customer_ip_location              6000 non-null   object
 9   card_expiration_delay             6000 non-null   int64
 10  payment_method                    6000 non-null   object
 11  login_status                      6000 non-null   object
 12  time_since_account_creation       6000 non-null   int64
 13  ip_address_previous_transactions  6000 non-null   int64
 14  user_segment                      2826 non-null   object
 15  product_views_during_session      5689 non-null   float64
 16  visit_origin                      5689 non-null   object
 17  device_type                       5689 non-null   object
dtypes: float64(3), int64(6), object(9)
memory usage: 843.9+ KB
None

Aperçu des données d'entraînement :
   transaction_id  cart_size  transaction_amount transaction_type  \
0         3196838          2                  98          product
1         3110633          1                  50          product
2         3001985          4                 220          product
3         3236215          1                  27          product
4         3260323          1                  43          product

  transaction_status customer_age_group  session_duration  \
0           complete              36-45              7.00
1           complete                NaN             11.31
2           complete                NaN             36.10
3           complete                NaN               NaN
4           complete              18-25             11.53

   time_spent_on_payment_page customer_ip_location  card_expiration_
delay  \
0                       24.79       non-eu/unknown
19
1                       41.66                local
19
2                       26.63                local
16
3                         NaN                local
15
4                         NaN                local
-1

  payment_method login_status  time_since_account_creation  \
0     creditcard    logged-in                          317
1     creditcard    logged-in                            0
2     creditcard    logged-in                          369
3     creditcard        guest                       -99999
4             4x    logged-in                           67

   ip_address_previous_transactions user_segment  \
0                                 0         AE1R
1                                 1          NaN
```

```
2                                  0         3R3T
3                                  2          NaN
4                                  0         NZ9Z

   product_views_during_session   visit_origin device_type  flag
0                           6.0           paid      mobile     0
1                           3.0  search engine      mobile     0
2                           4.0           paid      mobile     0
3                           NaN            NaN         NaN     1
4                           8.0          email      mobile     0

Aperçu des données de test :
   transaction_id  cart_size  transaction_amount transaction_type  \
0         3256667          1                  31          product
1         3164653          1                  66          product
2         3223879          2                 144          product
3         3101504          2                  57          product
4         3087301          2                 140          product

  transaction_status customer_age_group  session_duration  \
0           complete                NaN              6.80
1           complete                NaN             10.58
2           complete              26-35             23.02
3           complete                NaN             20.15
4           complete                NaN             16.82

   time_spent_on_payment_page customer_ip_location  card_expiration_
delay  \
0                         NaN                local
-1
1                         NaN                local
-1
2                       24.43                local
12
3                       30.69                local
19
4                         NaN                local
-1

  payment_method login_status  time_since_account_creation  \
0             4x    logged-in                          672
1      3rd party        guest                       -99999
2     creditcard    logged-in                          164
3     creditcard    logged-in                          377
4        instore        guest                       -99999

   ip_address_previous_transactions user_segment  \
0                                 0         NZ9Z
1                                 1          NaN
2                                 1         HQTL
3                                 0         T76E
4                                 1          NaN

   product_views_during_session   visit_origin device_type
0                           5.0           paid     desktop
1                           4.0           paid      mobile
```

```
2                          2.0  search engine     desktop
3                          2.0           email      mobile
4                          6.0  search engine      mobile
```

a. Description de la Base de Données

La base de données utilisée pour ce projet contient des informations détaillées sur les transactions réalisées sur une plateforme d'e-commerce. Elle est constituée de **19 variables** pour le jeu d'entraînement et **18 variables** pour le jeu de test, décrivant divers aspects des transactions. Ces variables incluent des caractéristiques liées aux comportements des utilisateurs, aux paiements, et aux contextes des transactions. La variable cible (`flag`) est binaire et indique si une transaction est légitime (**0**) ou frauduleuse (**1**).

Voici une description des variables principales :

- `transaction_id` : Identifiant unique de chaque transaction.
- `cart_size` : Nombre d'articles inclus dans la transaction.
- `transaction_amount` : Montant total de la transaction (en unités monétaires).
- `transaction_status` : Statut de la transaction (par exemple, "complete" ou "canceled").
- `transaction_type` : Type de la transaction (par exemple, "product" ou "service").
- `customer_age_group` : Groupe d'âge du client (par exemple, 18-25, 26-35).
- `session_duration` : Durée de la session utilisateur (en minutes).
- `time_spent_on_payment_page` : Temps passé sur la page de paiement (en secondes).
- `customer_ip_location` : Localisation globale de l'adresse IP du client.
- `card_expiration_delay` : Temps restant avant l'expiration de la carte utilisée (en mois).
- `payment_method` : Méthode de paiement utilisée (par exemple, carte de crédit, paiement en magasin).
- `login_status` : Statut de connexion de l'utilisateur (par exemple, "guest" ou "logged-in").
- `time_since_account_creation` : Temps écoulé depuis la création du compte utilisateur (en jours).
- `ip_address_previous_transactions` : Nombre de transactions précédentes effectuées à partir de la même adresse IP.
- `user_segment` : Segment utilisateur fourni par le département marketing.
- `product_views_during_session` : Nombre de produits visualisés

lors de la session.
- **`visit_origin`** : Origine du trafic web (par exemple, email, moteur de recherche).
- **`device_type`** : Type d'appareil utilisé (par exemple, mobile, desktop).
- **`flag`** : Indicateur binaire indiquant si la transaction est légitime (**0**) ou frauduleuse (**1**).

b. Structure de la Base de Données

Le jeu de données comprend des variables de types variés, qui permettent d'explorer plusieurs dimensions des transactions :

- **Variables numériques continues** :
  - `transaction_amount`, `session_duration`, `time_spent_on_payment_page`, `product_views_during_session`.
- **Variables numériques discrètes** :
  - `cart_size`, `card_expiration_delay`, `time_since_account_creation`, `ip_address_previous_transactions`.
- **Variables catégoriques** :
  - `transaction_type`, `transaction_status`, `customer_age_group`, `customer_ip_location`, `payment_method`, `login_status`, `user_segment`, `visit_origin`, `device_type`.
- **Variable cible** :
  - `flag` : Variable binaire utilisée pour indiquer si une transaction est frauduleuse.

La diversité des variables fournit un large éventail de caractéristiques pouvant aider à identifier des motifs de fraude potentiels. Les données présentent également des valeurs manquantes dans certaines colonnes, comme `customer_age_group` et `time_spent_on_payment_page`, nécessitant un prétraitement.

c. Analyse des variables et des Valeurs Manquantes

L'analyse des données révèle plusieurs points importants concernant les valeurs manquantes et la qualité des données :

1. **Variables avec des valeurs manquantes significatives** :
   - **`customer_age_group`** : Contient des valeurs manquantes

d'environ 63% des lignes pour le jeu d'entraînement et 64% pour le jeu de test.
    - **`time_spent_on_payment_page`** : Manque d'environ 34% des lignes du jeu d'entraînement et 33% du jeu de test.
    - **`user_segment`** : Près de 53% des valeurs manquent dans les jeux d'entraînement et test.
    - **`visit_origin`** et **`device_type`** : Présentent des valeurs manquantes d'environ 5.6% des lignes.
2. **Présence de valeurs aberrantes** :
    - Certaines variables, comme `time_since_account_creation`, contiennent des valeurs potentiellement erronées, comme `-99999`, qui devront être traitées.
3. **Densité de valeurs non nulles** :
    - Certaines colonnes, comme `transaction_id`, `cart_size`, et `flag`, ne présentent aucune valeur manquante, garantissant leur intégrité.
4. **Plan d'action pour les valeurs manquantes** :
    - Imputation des valeurs manquantes pour les variables numériques (`time_spent_on_payment_page`, `product_views_during_session`) en utilisant la moyenne ou la médiane.
    - Encodage des valeurs manquantes dans les variables catégoriques (`customer_age_group`, `user_segment`, `visit_origin`, `device_type`) avec une catégorie spéciale (par exemple, "inconnu").
    - Suppression ou traitement des lignes contenant des valeurs aberrantes.

d. Conclusion

Le jeu de données présente une richesse de caractéristiques permettant d'explorer des motifs de fraude potentiels. Cependant, une attention particulière doit être portée à la gestion des valeurs manquantes et des anomalies avant de procéder à l'analyse et à la modélisation. Une préparation rigoureuse des données garantira une meilleure performance du modèle et des résultats interprétables.

II. Exploration et Préparation des Données

L’exploration des données est une étape cruciale dans tout projet de data science, car elle permet de mieux comprendre la structure et les particularités du jeu de données. Cette phase vise à examiner les caractéristiques des variables, à identifier les tendances générales et les relations éventuelles entre les variables explicatives et la variable cible. Pour le jeu de données de détection de fraude, l’exploration des données inclura des analyses univariées et multivariées, permettant de visualiser la distribution des variables, de détecter des anomalies ou valeurs extrêmes, et de vérifier la présence de valeurs manquantes.

Cette analyse approfondie fournira ainsi les bases pour décider des étapes de prétraitement nécessaires, telles que le nettoyage des données, le formatage des variables et la transformation des caractéristiques. Une bonne compréhension des données dès le départ est essentielle pour construire des modèles robustes et efficaces pour détecter la fraude.

Nettoyage et formatage

1. Gestion des valeurs manquantes

Les valeurs manquantes, si elles ne sont pas traitées, peuvent introduire des biais ou réduire la représentativité des données. Voici les approches choisies pour gérer ces valeurs en fonction des types de variables :

- **Variables quantitatives** :
  - Les variables comme `session_duration` et `time_spent_on_payment_page` présentent des valeurs manquantes. Ces colonnes seront imputées :
    - Par la **moyenne**, si leur distribution est symétrique.
    - Par la **médiane**, si leur distribution contient des valeurs extrêmes.
- **Variables catégoriques** :
  - Les colonnes comme `customer_age_group` et `user_segment`, qui présentent un pourcentage significatif de valeurs manquantes, seront imputées par la valeur la plus fréquente (**mode**) ou une catégorie spécifique comme "inconnu".
- **Colonnes à taux élevé de valeurs manquantes** :
  - Si une colonne présente un trop grand nombre de valeurs manquantes (par exemple, plus de 50% des observations), elle sera envisagée pour suppression, sauf si elle a une importance stratégique.

2. Gestion des valeurs aberrantes

- **Variables numériques** :
  - Des valeurs extrêmes ont été identifiées, par exemple dans `time_since_account_creation` avec des valeurs comme `-99999`. Ces anomalies seront corrigées ou remplacées par des valeurs imputées (par exemple, la médiane).
- **Standardisation des variables** :
  - Les variables numériques comme `transaction_amount` seront standardisées pour garantir une distribution uniforme, facilitant l'entraînement des modèles.

3. Encodage des variables catégoriques

- Les variables catégoriques, telles que `transaction_type`, `visit_origin`, et `device_type`, seront transformées en variables numériques à l'aide de :
  - **Label Encoding** : Pour des catégories ordonnées.
  - **One-Hot Encoding** : Pour des catégories sans ordre, afin d'éviter toute confusion sur leur importance relative.

4. Suppression et création de nouvelles variables

- **Suppression des colonnes inutiles** :
  - Des colonnes non pertinentes pour la modélisation, comme `transaction_id`, seront exclues.
- **Création de nouvelles fonctionnalités** :
  - Générer des variables basées sur des combinaisons ou des transformations des données existantes, par exemple :
    - Le ratio entre `time_spent_on_payment_page` et `session_duration` pour évaluer l'engagement utilisateur.
    - Une variable binaire pour indiquer si une adresse IP est "répétitive" (basée sur `ip_address_previous_transactions`).

Mise en œuvre du nettoyage

Une fonction de nettoyage et d'imputation sera développer pour centraliser toutes les étapes décrites ci-dessus. Cette fonction appliquera les traitements suivants :

1. Remplacement des valeurs manquantes selon les stratégies définies.
2. Encodage des variables catégoriques en format numérique.

3. Suppression des colonnes non pertinentes ou excessivement manquantes.
4. Correction ou transformation des valeurs aberrantes.
5. Standardisation des variables numériques.

Cette méthodologie garantit la qualité et la cohérence des données, nécessaires pour une modélisation efficace.

In [38]:
```python
def clean_and_preprocess(data):
    """
    Nettoie et prépare les données pour la modélisation.

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données bru

    Returns:
        pd.DataFrame: Les données nettoyées et prêtes pour la modél
    """
    # Remplacement des valeurs aberrantes
    data['time_since_account_creation'] = data['time_since_account_

    # Imputation des valeurs manquantes (quantitatives)
    data['session_duration'] = data['session_duration'].fillna(data
    data['time_spent_on_payment_page'] = data['time_spent_on_paymen
    data['time_since_account_creation'] = data['time_since_account_
    data['product_views_during_session'] = data['product_views_duri

    # Imputation des valeurs manquantes (catégoriques)
    for col in ['customer_age_group', 'user_segment', 'visit_origin
        data[col] = data[col].fillna('inconnu')

    # Encodage des variables catégoriques
    categorical_cols = ['transaction_type', 'transaction_status', '
                        'customer_ip_location', 'payment_method', '
                        'visit_origin', 'device_type']
    data = pd.get_dummies(data, columns=categorical_cols, drop_firs

    # Standardisation des variables numériques
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    numeric_cols = ['transaction_amount', 'session_duration', 'time_
                    'time_since_account_creation', 'product_views_d
    data[numeric_cols] = scaler.fit_transform(data[numeric_cols])

    # Suppression des colonnes non pertinentes
    data = data.drop(columns=['transaction_id'])

    return data

# Application de la fonction de nettoyage sur les jeux de données
print("Nettoyage et préparation des données...")

train_data_cleaned = clean_and_preprocess(train_data)
test_data_cleaned = clean_and_preprocess(test_data)
```

```
print("Jeu de données d'entraînement nettoyé :")
print(train_data_cleaned.info())

print("\nJeu de données de test nettoyé :")
print(test_data_cleaned.info())

# Vérification des premières lignes après nettoyage
print("\nAperçu des données d'entraînement nettoyées :")
print(train_data_cleaned.head())

print("\nAperçu des données de test nettoyées :")
print(test_data_cleaned.head())
```

```
Nettoyage et préparation des données...
Jeu de données d'entraînement nettoyé :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24000 entries, 0 to 23999
Data columns (total 30 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   cart_size                             24000 non-null  int64  
 1   transaction_amount                    24000 non-null  float64
 2   session_duration                      24000 non-null  float64
 3   time_spent_on_payment_page            24000 non-null  float64
 4   card_expiration_delay                 24000 non-null  int64  
 5   time_since_account_creation           24000 non-null  float64
 6   ip_address_previous_transactions      24000 non-null  int64  
 7   user_segment                          24000 non-null  object 
 8   product_views_during_session          24000 non-null  float64
 9   flag                                  24000 non-null  int64  
 10  transaction_type_service              24000 non-null  bool   
 11  transaction_status_pending            24000 non-null  bool   
 12  customer_age_group_26-35              24000 non-null  bool   
 13  customer_age_group_36-45              24000 non-null  bool   
 14  customer_age_group_46-55              24000 non-null  bool   
 15  customer_age_group_56+                24000 non-null  bool   
 16  customer_age_group_inconnu            24000 non-null  bool   
 17  customer_ip_location_local            24000 non-null  bool   
 18  customer_ip_location_non-eu/unknown   24000 non-null  bool   
 19  payment_method_4x                     24000 non-null  bool   
 20  payment_method_creditcard             24000 non-null  bool   
 21  payment_method_instore                24000 non-null  bool   
 22  login_status_logged-in                24000 non-null  bool   
 23  visit_origin_email                    24000 non-null  bool   
 24  visit_origin_inconnu                  24000 non-null  bool   
 25  visit_origin_paid                     24000 non-null  bool   
 26  visit_origin_search engine            24000 non-null  bool   
 27  device_type_desktop                   24000 non-null  bool   
 28  device_type_inconnu                   24000 non-null  bool   
 29  device_type_mobile                    24000 non-null  bool   
dtypes: bool(20), float64(5), int64(4), object(1)
memory usage: 2.3+ MB
None

Jeu de données de test nettoyé :
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 29 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   cart_size                             6000 non-null   int64  
 1   transaction_amount                    6000 non-null   float64
 2   session_duration                      6000 non-null   float64
 3   time_spent_on_payment_page            6000 non-null   float64
 4   card_expiration_delay                 6000 non-null   int64  
 5   time_since_account_creation           6000 non-null   float64
 6   ip_address_previous_transactions      6000 non-null   int64  
 7   user_segment                          6000 non-null   object 
 8   product_views_during_session          6000 non-null   float64
 9   transaction_type_service              6000 non-null   bool   
 10  transaction_status_pending            6000 non-null   bool   
 11  customer_age_group_26-35              6000 non-null   bool   
 12  customer_age_group_36-45              6000 non-null   bool   
 13  customer_age_group_46-55              6000 non-null   bool   
 14  customer_age_group_56+                6000 non-null   bool   
 15  customer_age_group_inconnu            6000 non-null   bool   
 16  customer_ip_location_local            6000 non-null   bool   
 17  customer_ip_location_non-eu/unknown   6000 non-null   bool   
 18  payment_method_4x                     6000 non-null   bool   
 19  payment_method_creditcard             6000 non-null   bool   
 20  payment_method_instore                6000 non-null   bool   
 21  login_status_logged-in                6000 non-null   bool   
 22  visit_origin_email                    6000 non-null   bool   
 23  visit_origin_inconnu                  6000 non-null   bool   
 24  visit_origin_paid                     6000 non-null   bool   
 25  visit_origin_search engine            6000 non-null   bool   
 26  device_type_desktop                   6000 non-null   bool   
 27  device_type_inconnu                   6000 non-null   bool   
 28  device_type_mobile                    6000 non-null   bool   
dtypes: bool(20), float64(5), int64(3), object(1)
memory usage: 539.2+ KB
None

Aperçu des données d'entraînement nettoyées :
   cart_size  transaction_amount  session_duration  \
0          2            0.197816         -0.991201   
1          1           -0.737881         -0.549956   
2          4            2.576048          1.987969   
3          1           -1.186236         -0.263301   
4          1           -0.874337         -0.527433   

   time_spent_on_payment_page  card_expiration_delay  \
0               -1.235174e+00                     19   
1                2.400159e+00                     19   
2               -8.386709e-01                     16   
3                1.531156e-15                     15   
4                1.531156e-15                     -1   

   time_since_account_creation  ip_address_previous_transactions use
r_segment  \
0                    -0.475350                                  0   
```

```
AE1R
1                    -1.452910                                1
inconnu
2                    -0.314993                                0
3R3T
3                    -0.102211                                2
inconnu
4                    -1.246297                                0
NZ9Z

   product_views_during_session  flag  ...  payment_method_creditcar
d  \
0                      0.030589     0  ...                       Tru
e
1                     -1.324759     0  ...                       Tru
e
2                     -0.872976     0  ...                       Tru
e
3                      0.030589     1  ...                       Tru
e
4                      0.934155     0  ...                      Fals
e

   payment_method_instore  login_status_logged-in  visit_origin_emai
l  \
0                   False                    True               Fals
e
1                   False                    True               Fals
e
2                   False                    True               Fals
e
3                   False                   False               Fals
e
4                   False                    True                Tru
e

   visit_origin_inconnu  visit_origin_paid  visit_origin_search engi
ne  \
0                 False               True                        Fal
se
1                 False              False                         Tr
ue
2                 False               True                        Fal
se
3                  True              False                        Fal
se
4                 False              False                        Fal
se

   device_type_desktop  device_type_inconnu  device_type_mobile
0                False                False                True
1                False                False                True
2                False                False                True
3                False                 True               False
4                False                False                True
```

```
[5 rows x 30 columns]

Aperçu des données de test nettoyées :
   cart_size  transaction_amount  session_duration  \
0          1           -1.088494         -1.000520
1          1           -0.408837         -0.616302
2          2            1.105828          0.648162
3          2           -0.583606          0.356440
4          2            1.028153          0.017963

   time_spent_on_payment_page  card_expiration_delay  \
0                    0.000000                     -1
1                    0.000000                     -1
2                   -1.334961                     12
3                    0.052912                     19
4                    0.000000                     -1

   time_since_account_creation  ip_address_previous_transactions use
r_segment  \
0                     0.642104                                  0
NZ9Z
1                    -0.120584                                  1
inconnu
2                    -0.895378                                  1
HQTL
3                    -0.250726                                  0
T76E
4                    -0.120584                                  1
inconnu

   product_views_during_session  transaction_type_service  ...  \
0                     -0.421769                     False  ...
1                     -0.877408                     False  ...
2                     -1.788685                     False  ...
3                     -1.788685                     False  ...
4                      0.033869                     False  ...

   payment_method_creditcard  payment_method_instore  login_status_l
ogged-in  \
0                      False                   False
True
1                      False                   False
False
2                       True                   False
True
3                       True                   False
True
4                      False                    True
False

   visit_origin_email  visit_origin_inconnu  visit_origin_paid  \
0               False                 False               True
1               False                 False               True
2               False                 False              False
3                True                 False              False
4               False                 False              False
```

```
   visit_origin_search engine  device_type_desktop  device_type_inco
nnu  \
0                       False                 True                Fa
lse
1                       False                False                Fa
lse
2                        True                 True                Fa
lse
3                       False                False                Fa
lse
4                        True                False                Fa
lse

   device_type_mobile
0               False
1                True
2               False
3                True
4                True

[5 rows x 29 columns]
```

Les données ont été soigneusement nettoyées et préparées pour la modélisation. Cela inclut l'imputation des valeurs manquantes pour les variables quantitatives et catégoriques, le traitement des valeurs aberrantes comme `-99999` dans la colonne `time_since_account_creation`, ainsi que la standardisation des variables numériques pour uniformiser leurs échelles. Les variables catégoriques ont été transformées en variables binaires via un encodage **One-Hot**, augmentant le nombre total de colonnes tout en rendant les données exploitables par les modèles. Ce processus a permis de transformer les données brutes en un ensemble structuré, complet et optimisé, prêt pour l'analyse exploratoire et la modélisation.

Pour assurer une compatibilité universelle avec les algorithmes de machine learning, il est important de convertir les variables booléennes (`True/False`) en format numérique (`0/1`). Cette étape garantit une homogénéité dans le type de données tout en évitant d'éventuels problèmes liés au traitement des variables non numériques. Voici le code développé pour effectuer cette conversion sur les jeux de données nettoyés.

In [39]:
```
def convert_boolean_to_numeric(data):
    """
    Convertit toutes les colonnes de type booléen en numérique (0 p

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données net

    Returns:
        pd.DataFrame: Les données avec les colonnes booléennes conv
    """
    # Identifier les colonnes booléennes
```

```
    boolean_columns = data.select_dtypes(include='bool').columns
    print(f"Colonnes booléennes détectées ({len(boolean_columns)}):

    # Conversion en numérique
    data[boolean_columns] = data[boolean_columns].astype(int)
    return data

# Application sur les jeux de données nettoyés
train_data_cleaned = convert_boolean_to_numeric(train_data_cleaned)
test_data_cleaned = convert_boolean_to_numeric(test_data_cleaned)

# Vérification après conversion
print(train_data_cleaned.info())
print(test_data_cleaned.info())
```

```
Colonnes booléennes détectées (20): ['transaction_type_service', 'tr
ansaction_status_pending', 'customer_age_group_26-35', 'customer_age
_group_36-45', 'customer_age_group_46-55', 'customer_age_group_56+',
'customer_age_group_inconnu', 'customer_ip_location_local', 'custome
r_ip_location_non-eu/unknown', 'payment_method_4x', 'payment_method_
creditcard', 'payment_method_instore', 'login_status_logged-in', 'vi
sit_origin_email', 'visit_origin_inconnu', 'visit_origin_paid', 'vis
it_origin_search engine', 'device_type_desktop', 'device_type_inconn
u', 'device_type_mobile']
Colonnes booléennes détectées (20): ['transaction_type_service', 'tr
ansaction_status_pending', 'customer_age_group_26-35', 'customer_age
_group_36-45', 'customer_age_group_46-55', 'customer_age_group_56+',
'customer_age_group_inconnu', 'customer_ip_location_local', 'custome
r_ip_location_non-eu/unknown', 'payment_method_4x', 'payment_method_
creditcard', 'payment_method_instore', 'login_status_logged-in', 'vi
sit_origin_email', 'visit_origin_inconnu', 'visit_origin_paid', 'vis
it_origin_search engine', 'device_type_desktop', 'device_type_inconn
u', 'device_type_mobile']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24000 entries, 0 to 23999
Data columns (total 30 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   cart_size                             24000 non-null  int64
 1   transaction_amount                    24000 non-null  float64
 2   session_duration                      24000 non-null  float64
 3   time_spent_on_payment_page            24000 non-null  float64
 4   card_expiration_delay                 24000 non-null  int64
 5   time_since_account_creation           24000 non-null  float64
 6   ip_address_previous_transactions      24000 non-null  int64
 7   user_segment                          24000 non-null  object
 8   product_views_during_session          24000 non-null  float64
 9   flag                                  24000 non-null  int64
 10  transaction_type_service              24000 non-null  int64
 11  transaction_status_pending            24000 non-null  int64
 12  customer_age_group_26-35              24000 non-null  int64
 13  customer_age_group_36-45              24000 non-null  int64
 14  customer_age_group_46-55              24000 non-null  int64
 15  customer_age_group_56+                24000 non-null  int64
 16  customer_age_group_inconnu            24000 non-null  int64
 17  customer_ip_location_local            24000 non-null  int64
```

```
 18  customer_ip_location_non-eu/unknown  24000 non-null  int64
 19  payment_method_4x                    24000 non-null  int64
 20  payment_method_creditcard            24000 non-null  int64
 21  payment_method_instore               24000 non-null  int64
 22  login_status_logged-in               24000 non-null  int64
 23  visit_origin_email                   24000 non-null  int64
 24  visit_origin_inconnu                 24000 non-null  int64
 25  visit_origin_paid                    24000 non-null  int64
 26  visit_origin_search engine           24000 non-null  int64
 27  device_type_desktop                  24000 non-null  int64
 28  device_type_inconnu                  24000 non-null  int64
 29  device_type_mobile                   24000 non-null  int64
dtypes: float64(5), int64(24), object(1)
memory usage: 5.5+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 29 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   cart_size                            6000 non-null   int64
 1   transaction_amount                   6000 non-null   float64
 2   session_duration                     6000 non-null   float64
 3   time_spent_on_payment_page           6000 non-null   float64
 4   card_expiration_delay                6000 non-null   int64
 5   time_since_account_creation          6000 non-null   float64
 6   ip_address_previous_transactions     6000 non-null   int64
 7   user_segment                         6000 non-null   object
 8   product_views_during_session         6000 non-null   float64
 9   transaction_type_service             6000 non-null   int64
 10  transaction_status_pending           6000 non-null   int64
 11  customer_age_group_26-35             6000 non-null   int64
 12  customer_age_group_36-45             6000 non-null   int64
 13  customer_age_group_46-55             6000 non-null   int64
 14  customer_age_group_56+               6000 non-null   int64
 15  customer_age_group_inconnu           6000 non-null   int64
 16  customer_ip_location_local           6000 non-null   int64
 17  customer_ip_location_non-eu/unknown  6000 non-null   int64
 18  payment_method_4x                    6000 non-null   int64
 19  payment_method_creditcard            6000 non-null   int64
 20  payment_method_instore               6000 non-null   int64
 21  login_status_logged-in               6000 non-null   int64
 22  visit_origin_email                   6000 non-null   int64
 23  visit_origin_inconnu                 6000 non-null   int64
 24  visit_origin_paid                    6000 non-null   int64
 25  visit_origin_search engine           6000 non-null   int64
 26  device_type_desktop                  6000 non-null   int64
 27  device_type_inconnu                  6000 non-null   int64
 28  device_type_mobile                   6000 non-null   int64
dtypes: float64(5), int64(23), object(1)
memory usage: 1.3+ MB
None
```

Les colonnes booléennes initiales (`True/False`) ont été converties avec succès en variables numériques (`0/1`), garantissant une compatibilité universelle avec les algorithmes de machine learning. Un total de 20 variables

ont été identifiées et transformées, incluant des variables comme `transaction_type_service`, `payment_method_creditcard`, et `device_type_mobile`. Désormais, les données contiennent principalement des types numériques (`int64`, `float64`), avec une seule variable restante de type `object` (`user_segment`). Cette homogénéité des types de données optimise la structure du jeu de données, le rendant totalement prêt pour les étapes d'analyse exploratoire et de modélisation.

III. Analyse descriptive des variables et exploration des données pour la détection de fraude

Pour mieux comprendre les caractéristiques du jeu de données et les relations potentielles entre les variables explicatives et la variable cible, nous effectuons une analyse descriptive comprenant une visualisation univariée et multivariée. L'analyse univariée permet d'explorer la distribution de chaque variable individuellement, tandis que l'analyse multivariée aide à identifier les corrélations ou interactions entre les variables. Ces visualisations offrent une vue d'ensemble des données et des patterns sous-jacents, facilitant ainsi les décisions futures pour la modélisation et l'interprétation des résultats.

1. Étude univariée

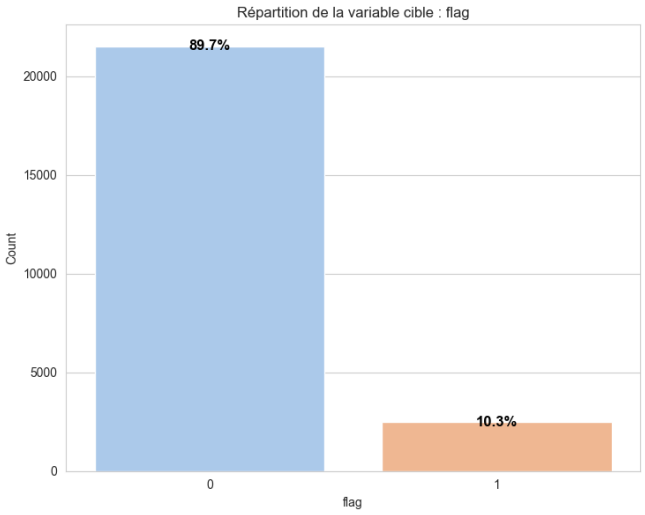
1.1 Variable dépendante (Variable Cible)

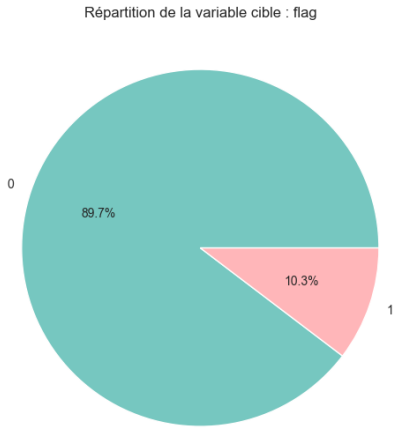
Dans cette section, nous analysons la variable cible de notre projet : **la détection de fraude** (`flag`). Cette variable binaire (0 pour une transaction normale et 1 pour une transaction frauduleuse) est essentielle pour notre objectif de classification. Comprendre la distribution de cette variable est crucial, car elle détermine l'équilibre entre les classes et influence le choix des métriques d'évaluation des modèles. Une analyse univariée de la variable cible permettra d'identifier la proportion de transactions frauduleuses par rapport aux transactions normales, mettant en lumière un éventuel déséquilibre des classes qui pourrait nécessiter une stratégie d'ajustement comme un suréchantillonnage ou un sous-échantillonnage. Cela servira également de base pour l'évaluation des performances du modèle.

In [40]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Définition des colonnes numériques et catégorielles après nettoya
numerical_columns = ['transaction_amount', 'session_duration', 'tim
                     'time_since_account_creation', 'product_views_
```

```
categorical_columns = ['transaction_type_service', 'transaction_sta
                       'customer_age_group_26-35', 'customer_age_gr
                       'customer_age_group_56+', 'customer_age_grou
                       'customer_ip_location_local', 'customer_ip_l
                       'payment_method_4x', 'payment_method_creditc
                       'login_status_logged-in', 'visit_origin_emai
                       'visit_origin_paid', 'visit_origin_search en
                       'device_type_inconnu', 'device_type_mobile']


def univariate_visualization(data, numerical_columns, categorical_c
    """
    Affiche les visualisations univariées pour chaque variable quan
    Si une variable cible est spécifiée, elle sera également visual
    """
    if categorical_columns is None:
        categorical_columns = []

    # Visualisation de la variable cible binaire uniquement si targ
    if target_variable:
        # Création de la figure avec 2 sous-graphes (countplot et p
        fig, axes = plt.subplots(1, 2, figsize=(14, 6))  # 1 ligne,

        # Countplot pour la variable binaire (affichage sur le prem
        sns.countplot(data=data, x=target_variable, palette="pastel"
        axes[0].set_title(f"Répartition de la variable cible : {tar
        axes[0].set_xlabel(target_variable)
        axes[0].set_ylabel('Count')

        # Ajout des pourcentages sur chaque barre du countplot
        total = len(data[target_variable])
        for p in axes[0].patches:
            height = p.get_height()
            percentage = (height / total) * 100
            axes[0].annotate(f'{percentage:.1f}%', (p.get_x() + p.g
                             ha='center', va='center', fontsize=12,

        # Diagramme de Camembert pour la variable cible (affichage
        data[target_variable].value_counts().plot(kind='pie', autop
        axes[1].set_title(f"Répartition de la variable cible : {tar
        axes[1].set_ylabel('')  # Enlève le label de l'axe y

        # Affichage des graphiques
        plt.tight_layout()  # Assure une meilleure disposition
        plt.show()

    # Visualisation des variables quantitatives si target_variable
    if len(numerical_columns) > 0 and not target_variable:
        plt.figure(figsize=(15, 10))
        for i, col in enumerate(numerical_columns):
            color = sns.color_palette('coolwarm')[2]

            # Histogramme
            plt.subplot(len(numerical_columns), 2, 2 * i + 1)
            sns.histplot(data[col], kde=True, bins=30, color=color)
```

```
            plt.title(f"Distribution de {col}", fontsize=12)
            plt.xlabel(col, fontsize=10)
            plt.ylabel("Count", fontsize=10)

            # Boxplot
            plt.subplot(len(numerical_columns), 2, 2 * i + 2)
            sns.boxplot(x=data[col], color=color)
            plt.title(f"Boxplot de {col}", fontsize=12)
            plt.xlabel(col, fontsize=10)

        plt.tight_layout()
        plt.show()

    # Visualisation des variables catégorielles si target_variable (
    if len(categorical_columns) > 0 and not target_variable:
        plt.figure(figsize=(15, 10))
        for i, col in enumerate(categorical_columns):
            plt.subplot(len(categorical_columns), 1, i + 1)
            sns.countplot(data=data, x=col, palette="muted")
            plt.title(f"Distribution de {col}", fontsize=12)
            plt.xlabel(col, fontsize=10)
            plt.ylabel("Count", fontsize=10)

        plt.tight_layout()
        plt.show()


# Appliquer la fonction pour la variable cible 'flag'
univariate_visualization(train_data_cleaned, numerical_columns, cat
```



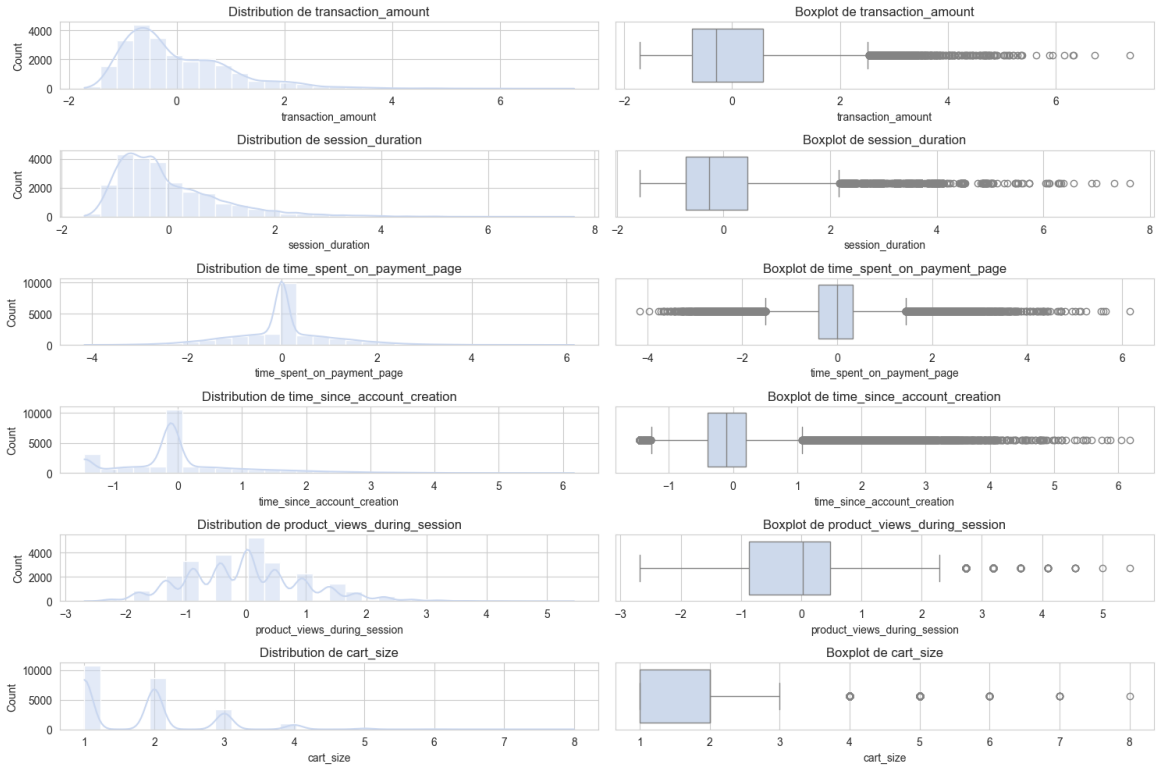


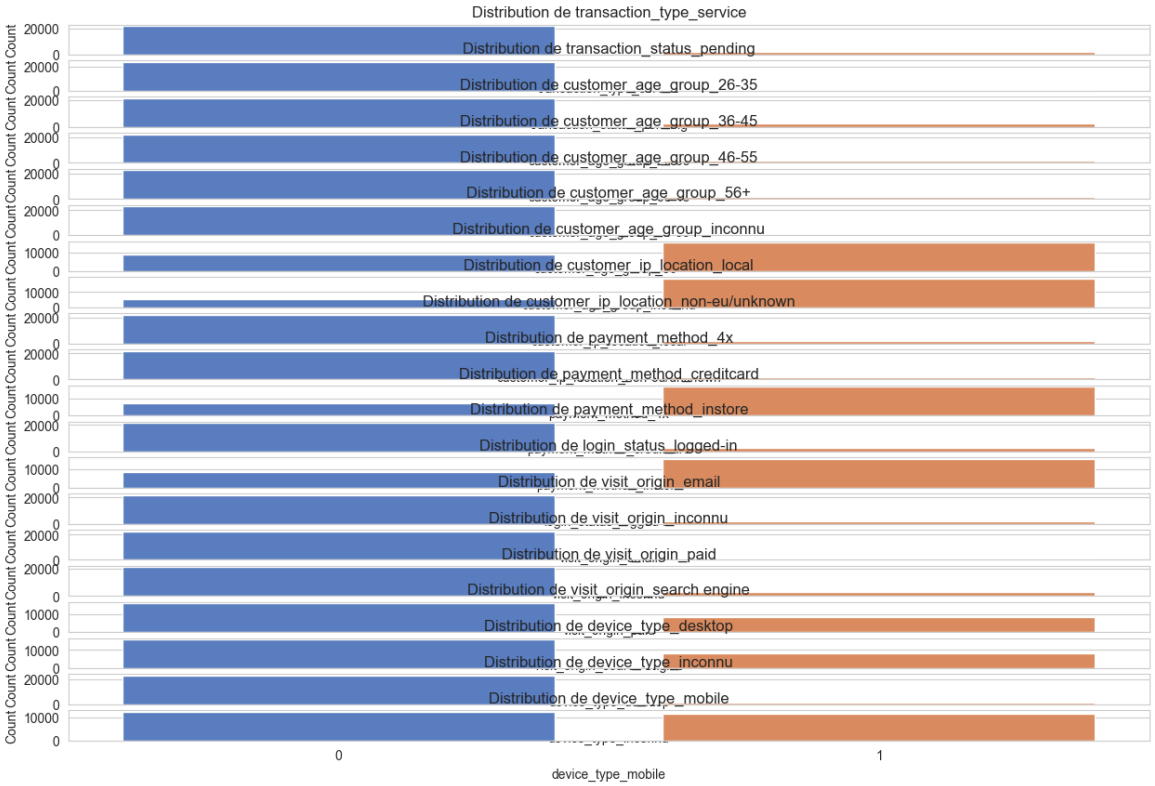
L'analyse de la variable cible $flag$ montre un fort déséquilibre entre ses deux classes. La classe 0, qui représente les transactions normales, constitue environ $89,7\%$ des observations, tandis que la classe 1, qui représente les transactions suspectes, n'en constitue que d'environ $10,3\%$. Ce déséquilibre est clairement visible dans le graphique à barres et le diagramme circulaire, où la classe 0 domine largement.

1.2. Variables explicatives

Les variables explicatives sont au nombre de 30 : Cinq variables continues ($transaction\_amount$, $session\_duration$, $time\_spent\_on\_payment\_page$, $time\_since\_account\_creation$, $product\_views\_during\_session$) et 25 variables catégorielles, dont 20 ont été converties en variables binaires à l'aide de l'encodage **One-Hot** ($transaction\_type$, $transaction\_status$, $customer\_age\_group$, $customer\_ip\_location$, $payment\_method$, $login\_status$, $visit\_origin$, $device\_type$, $user\_segment$, $ip\_address\_previous\_transactions$, $card\_expiration\_delay$).

In [41]:
```
# Application des statistiques univariées et les visualisations pou
univariate_visualization(train_data_cleaned, numerical_columns, cat
```





a. Analyse et Interprétation des Sorties

1. **Variables Quantitatives :**

- **Montant de la transaction (`transaction_amount`)** : La distribution de cette variable montre une forte asymétrie, avec des montants plus faibles dominants, mais quelques transactions atteignant des valeurs exceptionnellement élevées. Le boxplot met en évidence plusieurs valeurs aberrantes, ce qui pourrait être lié à des transactions frauduleuses ou exceptionnelles.
- **Durée de la session (`session_duration`)** : La majorité des utilisateurs semblent passer un temps relativement court sur la plateforme, ce qui est visible dans la concentration des valeurs autour des petites durées. Cependant, des valeurs extrêmes suggèrent que certains utilisateurs peuvent interagir avec la plateforme pendant une période anormalement longue, ce qui pourrait être associé à des tentatives de fraude.
- **Temps passé sur la page de paiement (`time_spent_on_payment_page`)** : Cette variable présente une forte concentration de petites valeurs, indiquant que la plupart des utilisateurs passent peu de temps sur la page de paiement. Toutefois, quelques valeurs aberrantes sont présentes, ce qui pourrait signifier des comportements suspects comme des tentatives de manipulation des informations de paiement.
- **Temps depuis la création du compte (`time_since_account_creation`)** : La majorité des utilisateurs ont créé leur compte récemment, comme le montre la concentration autour

de faibles valeurs. Cependant, certaines valeurs extrêmes, particulièrement les valeurs négatives ou proches de zéro, peuvent être liées à des tentatives de fraude par usurpation d'identité ou manipulation du système.
- **Vues de produits pendant la session (`product_views_during_session`)** : Les utilisateurs passent en moyenne peu de temps à explorer les produits, mais il existe quelques valeurs extrêmes, ce qui peut indiquer des comportements suspects tels que des achats en masse ou des tentatives de fraude par exploration automatisée des produits.

2. **Variables Catégorielles :**

- **Type de transaction (`transaction_type`)** : La majorité des transactions sont de type "produit", mais un nombre significatif concerne des "services". Ce type de transaction peut être plus difficile à surveiller et pourrait représenter un vecteur pour des fraudes. Il est important d'analyser plus en profondeur ce facteur pour identifier des motifs de fraude.
- **Statut de la transaction (`transaction_status`)** : La majorité des transactions ont un statut "complété", mais une part notable est en "attente". Cela pourrait refléter des tentatives de fraude où la transaction est en pause ou en cours de validation. Une attention particulière devrait être accordée aux transactions en attente.
- **Méthode de paiement (`payment_method`)** : La majorité des utilisateurs utilisent des cartes de crédit pour effectuer leurs achats, mais il existe des utilisateurs qui préfèrent d'autres méthodes de paiement, comme "instore" ou "4x". Ces méthodes peuvent être plus risquées et nécessitent une surveillance particulière pour détecter des transactions frauduleuses.
- **Statut de connexion (`login_status`)** : Un nombre important d'utilisateurs sont connectés à leur compte lors des transactions, tandis qu'un petit pourcentage se connecte en tant qu'invité. Les utilisateurs invités, notamment s'ils effectuent des transactions importantes ou multiples, pourraient représenter des comportements suspects.
- **Origine de la visite (`visit_origin`)** : Cette variable montre que la majorité des visites proviennent de sources comme "paid" et "search engine". Ces canaux peuvent être des points d'entrée importants pour des tentatives de fraude, par exemple, des publicités frauduleuses ou des moteurs de recherche utilisés pour rediriger vers des pages non sécurisées.
- **Type de dispositif (`device_type`)** : Les utilisateurs accèdent principalement au site via des appareils mobiles, mais un nombre important utilise aussi des ordinateurs de bureau. Il est essentiel de vérifier si les comportements suspects sont associés à certains types

d'appareils, par exemple, si les utilisateurs mobiles sont plus susceptibles de frauder.

3. **Autres Variables :**

- `customer_age_group` : La répartition par groupe d'âge montre une prédominance des utilisateurs dans les tranches d'âge 26-35, suivie par les autres groupes. Une analyse plus approfondie peut révéler si certains groupes d'âge sont plus enclins à effectuer des transactions frauduleuses.
- `customer_ip_location` : Cette variable indique que la majorité des utilisateurs viennent de localisations connues (local), tandis que quelques-uns proviennent de régions inconnues ou non-européennes. Les utilisateurs provenant de régions inconnues peuvent nécessiter une attention particulière, car cela pourrait indiquer un comportement de fraude, en particulier si ces utilisateurs ont un historique de transactions suspectes.
- `user_segment` : La majorité des utilisateurs sont regroupés dans un segment spécifique, mais il existe plusieurs segments avec des tailles différentes. Certains segments plus petits pourraient avoir des comportements anormaux et nécessitent une attention particulière.

b. Conclusion

L'analyse des données montre une grande diversité dans les comportements des utilisateurs et les caractéristiques des transactions. Les variables quantitatives, comme le montant des transactions et le temps passé sur la page de paiement, présentent des valeurs extrêmes qui méritent d'être surveillées de près pour détecter des anomalies ou des fraudes. Les variables catégorielles révèlent également des déséquilibres, en particulier pour les méthodes de paiement, le statut de la transaction, et l'origine de la visite. Les utilisateurs invités et les transactions en attente semblent être des indicateurs potentiels de fraude.

Certaines variables, comme le statut de connexion et l'origine de la visite, révèlent des indices potentiels de fraude, en particulier les utilisateurs invités et les transactions en attente. Une attention particulière doit être portée à ces utilisateurs pour déterminer si leurs actions sont légitimes ou suspectes.

Enfin, les valeurs aberrantes dans les variables quantitatives doivent être traitées avant de passer à l'étape de modélisation. La gestion des utilisateurs provenant de zones géographiques inconnues et des utilisateurs invités pourrait également être un facteur clé dans la détection de la fraude.

1.3. Traitement des valeurs Abérantes et des outliers

a. Identification des Outliers

Dans cette section, nous allons détecter les **valeurs aberrantes** présentes dans les variables numériques du jeu de données en utilisant la méthode de l'**IQR (Interquartile Range)**. Cette approche statistique permettra d'identifier les valeurs extrêmes situées en dehors de l'intervalle $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$, où $Q1$ et $Q3$ sont respectivement les 1^{er} et 3^{e} quartiles, et $IQR = Q3 - Q1$.

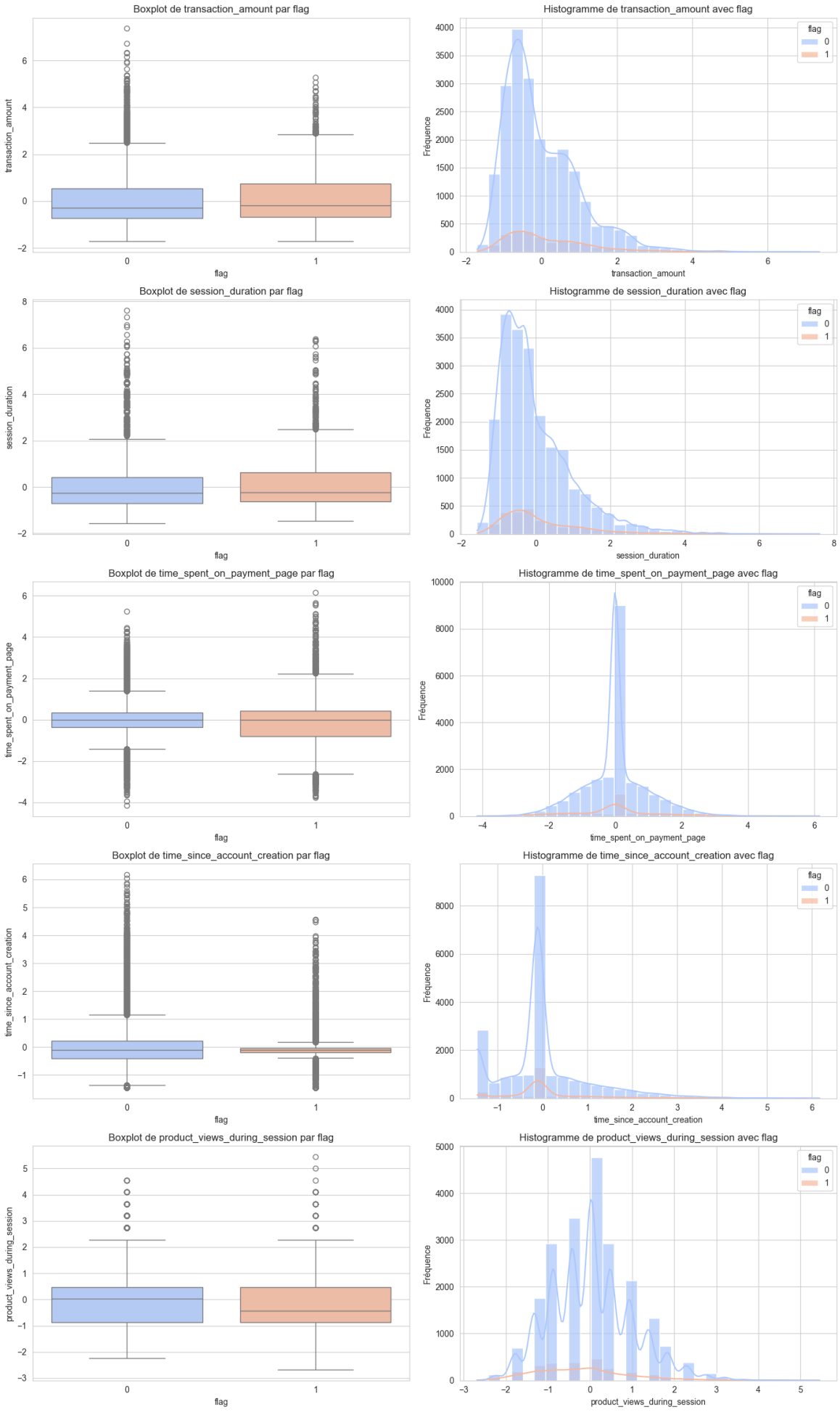
Une fois les valeurs aberrantes détectées, nous analyserons leur distribution et leur lien potentiel avec la variable cible $flag$. Cela nous permettra de comprendre si ces valeurs extrêmes contiennent des informations pertinentes pour la modélisation ou si elles nécessitent un traitement spécifique pour garantir la qualité et la performance des modèles prédictifs.

In [42]:
```
def detect_outliers_and_flag(data, columns):
    """
    Détecte les valeurs aberrantes avec l'IQR(Interquartile Range) 

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données.
        columns (list): Liste des colonnes numériques à analyser.

    Returns:
        dict: Dictionnaire contenant les indices des outliers et le
    """
    outliers_info = {}
    for col in columns:
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Détection des outliers
        outliers = data[(data[col] < lower_bound) | (data[col] > up

        # Lien avec la variable cible
        fraud_related = outliers['flag'].value_counts(normalize=Tru

        outliers_info[col] = {
            "indices": outliers.index.tolist(),
            "fraud_related": fraud_related
        }

    return outliers_info

# Colonnes à analyser
numeric_columns = ['transaction_amount', 'session_duration',
                   'time_spent_on_payment_page', 'time_since_accoun
                   'product_views_during_session']

outliers_details = detect_outliers_and_flag(train_data_cleaned, num
```

Visualisation des Outliers

In [43]:
```
import seaborn as sns
import matplotlib.pyplot as plt

def plot_outliers_with_target_combined(data, columns, target='flag'
    """
    Affiche les boxplots et histogrammes des colonnes en relation a

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données.
        columns (list): Liste des colonnes à analyser.
        target (str): La variable cible.

    Returns:
        None
    """
    n_cols = len(columns)
    fig, axes = plt.subplots(n_cols, 2, figsize=(14, 5 * n_cols))
    fig.suptitle("Visualisation des valeurs aberrantes par variable

    for i, col in enumerate(columns):
        # Boxplot (colonne de gauche)
        sns.boxplot(x=target, y=col, data=data, ax=axes[i, 0], pale
        axes[i, 0].set_title(f"Boxplot de {col} par {target}")
        axes[i, 0].set_xlabel(target)
        axes[i, 0].set_ylabel(col)

        # Histogramme (colonne de droite)
        sns.histplot(data=data, x=col, hue=target, kde=True, ax=axe
        axes[i, 1].set_title(f"Histogramme de {col} avec {target}")
        axes[i, 1].set_xlabel(col)
        axes[i, 1].set_ylabel("Fréquence")

    plt.tight_layout(rect=[0, 0, 1, 0.95])  # Ajuste les sous-graph
    plt.show()

# Colonnes numériques à analyser
numeric_columns = ['transaction_amount', 'session_duration',
                   'time_spent_on_payment_page', 'time_since_accoun
                   'product_views_during_session']

# Appel de la fonction pour visualiser les boxplots et histogrammes
plot_outliers_with_target_combined(train_data_cleaned, numeric_colu
```

Visualisation des valeurs aberrantes par variable cible



Analyse des résultats

1. **Montant de la transaction (`transaction_amount`)**

- Les transactions frauduleuses (`flag = 1`) présentent une distribution légèrement plus élevée que les transactions non frauduleuses (`flag = 0`).
- Cependant, les deux classes montrent un grand nombre de valeurs aberrantes vers le haut, indiquant des montants très élevés.
- **Interprétation** : Les valeurs aberrantes pour cette variable peuvent être informatives dans la détection de fraudes. Ces montants élevés doivent être conservés pour la modélisation.

2. **Durée de la session (`session_duration`)**

- Les deux classes montrent des valeurs extrêmes similaires pour cette variable, avec une majorité de données concentrées sur des durées courtes.
- Les valeurs aberrantes plus longues ne semblent pas différencier fortement les transactions frauduleuses des transactions normales.
- **Interprétation** : Les valeurs extrêmes peuvent être imputées ou normalisées, car elles n'apportent pas une information claire liée à la fraude.

3. **Temps passé sur la page de paiement (`time_spent_on_payment_page`)**

- Les distributions sont similaires pour les deux classes. Les valeurs aberrantes (temps très long ou très court) ne semblent pas corrélées avec le drapeau de fraude.
- **Interprétation** : Ces valeurs peuvent être traitées pour réduire leur impact sur la modélisation, soit par capping, soit par transformation logarithmique.

4. **Temps depuis la création du compte (`time_since_account_creation`)**

- Les fraudes (`flag = 1`) semblent légèrement plus concentrées sur les comptes récents, mais les valeurs aberrantes (comptes anciens ou créés récemment avec des valeurs négatives) apparaissent dans les deux classes.
- **Interprétation** : Les valeurs aberrantes ici, notamment les valeurs négatives, doivent être corrigées ou imputées.

5. **Vues de produits pendant la session (`product_views_during_session`)**

- Les distributions sont similaires entre les deux classes. Les valeurs aberrantes (sessions avec un très grand nombre de vues) ne semblent pas fortement liées à la fraude.
- **Interprétation** : Ces valeurs peuvent être traitées pour réduire leur impact sur les modèles.

b. Gestion des Valeurs Aberrantes et Outliers

1. **Conservation des valeurs aberrantes informatives :**

Les valeurs extrêmes de certaines variables, telles que `transaction_amount`, contiennent des informations précieuses pour la détection des fraudes. Ces valeurs seront conservées dans le jeu de données, car elles peuvent refléter des comportements suspects ou des anomalies significatives.

2. **Limitation des valeurs extrêmes (Capping) ou transformation logarithmique :**

Pour des variables telles que `session_duration`, `time_spent_on_payment_page`, et `product_views_during_session`, les valeurs aberrantes peuvent perturber la modélisation sans fournir d'informations spécifiques liées à la fraude. Afin de réduire leur impact, une **transformation logarithmique** sera appliquée pour atténuer l'effet des valeurs élevées.

3. **Imputation des valeurs irrationnelles :**

Les valeurs aberrantes irrationnelles ou incohérentes, telles que les valeurs négatives dans `time_since_account_creation`, seront corrigées. Ces valeurs seront imputées à l'aide de la médiane ou d'autres mesures statistiques afin de garantir la cohérence et la qualité des données.

Résumé de la Gestion des Outliers et des Valeurs Aberrantes

Variable	Nature des Valeurs	Traitement
`transaction_amount`	Outliers	Conserver pour la modélisation, appliquer une transformation logarithmique.
`session_duration`	Outliers	Appliquer une transformation logarithmique pour réduire l'impact des valeurs extrêmes.

`time_spent_on_payment_page`	Outliers	Appliquer une transformation logarithmique ou un capping des valeurs élevées.
`time_since_account_creation`	Outliers + Valeurs aberrantes	Imputer les valeurs aberrantes (négatives) avec la médiane et appliquer une transformation logarithmique pour les autres.
`product_views_during_session`	Outliers	Appliquer une transformation logarithmique ou un capping pour réduire l'impact des valeurs extrêmes.

c. Visualisation des distributions après traitement

In [44]:

```python
def process_outliers_combined(data, columns, target='flag'):
    """
    Traite les outliers en appliquant capping et transformation log
    tout en conservant le lien avec les classes de la variable cibl

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données.
        columns (list): Liste des colonnes numériques à traiter.
        target (str): La variable cible pour l'analyse des classes

    Returns:
        pd.DataFrame: Le DataFrame avec les valeurs traitées.
    """
    for col in columns:
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Affichage des informations sur les outliers
        print(f"Analyse des outliers pour '{col}':")
        outliers = data[(data[col] < lower_bound) | (data[col] > up
        if not outliers.empty:
            print(f"Nombre d'outliers : {len(outliers)}")
            print(f"Répartition des classes :\n{outliers[target].va
        else:
            print("Aucun outlier détecté.\n")

        # Capping des valeurs extrêmes
        data[col] = data[col].clip(lower=lower_bound, upper=upper_b

        # Transformation logarithmique après capping
        data[col] = np.log1p(data[col].clip(lower=0))

        # Imputation des valeurs négatives pour 'time_since_account
```

```
        if col == 'time_since_account_creation':
            median_value = data[col].median()
            data[col] = data[col].apply(lambda x: median_value if x

    return data

# Liste des colonnes à traiter
columns_to_process = ['transaction_amount', 'session_duration',
                      'time_spent_on_payment_page', 'time_since_acc
                      'product_views_during_session']

# Application sur les données
print("Traitement des outliers avec transformations en cours...")
train_data_cleaned = process_outliers_combined(train_data_cleaned,

# Vérification finale
print("\nAperçu des données après traitement :")
print(train_data_cleaned.head())
```

```
Traitement des outliers avec transformations en cours...
Analyse des outliers pour 'transaction_amount':
Nombre d'outliers : 529
Répartition des classes :
flag
0    0.824197
1    0.175803
Name: proportion, dtype: float64

Analyse des outliers pour 'session_duration':
Nombre d'outliers : 947
Répartition des classes :
flag
0    0.850053
1    0.149947
Name: proportion, dtype: float64

Analyse des outliers pour 'time_spent_on_payment_page':
Nombre d'outliers : 3547
Répartition des classes :
flag
0    0.803778
1    0.196222
Name: proportion, dtype: float64

Analyse des outliers pour 'time_since_account_creation':
Nombre d'outliers : 5963
Répartition des classes :
flag
0    0.914473
1    0.085527
Name: proportion, dtype: float64

Analyse des outliers pour 'product_views_during_session':
Nombre d'outliers : 307
Répartition des classes :
flag
```

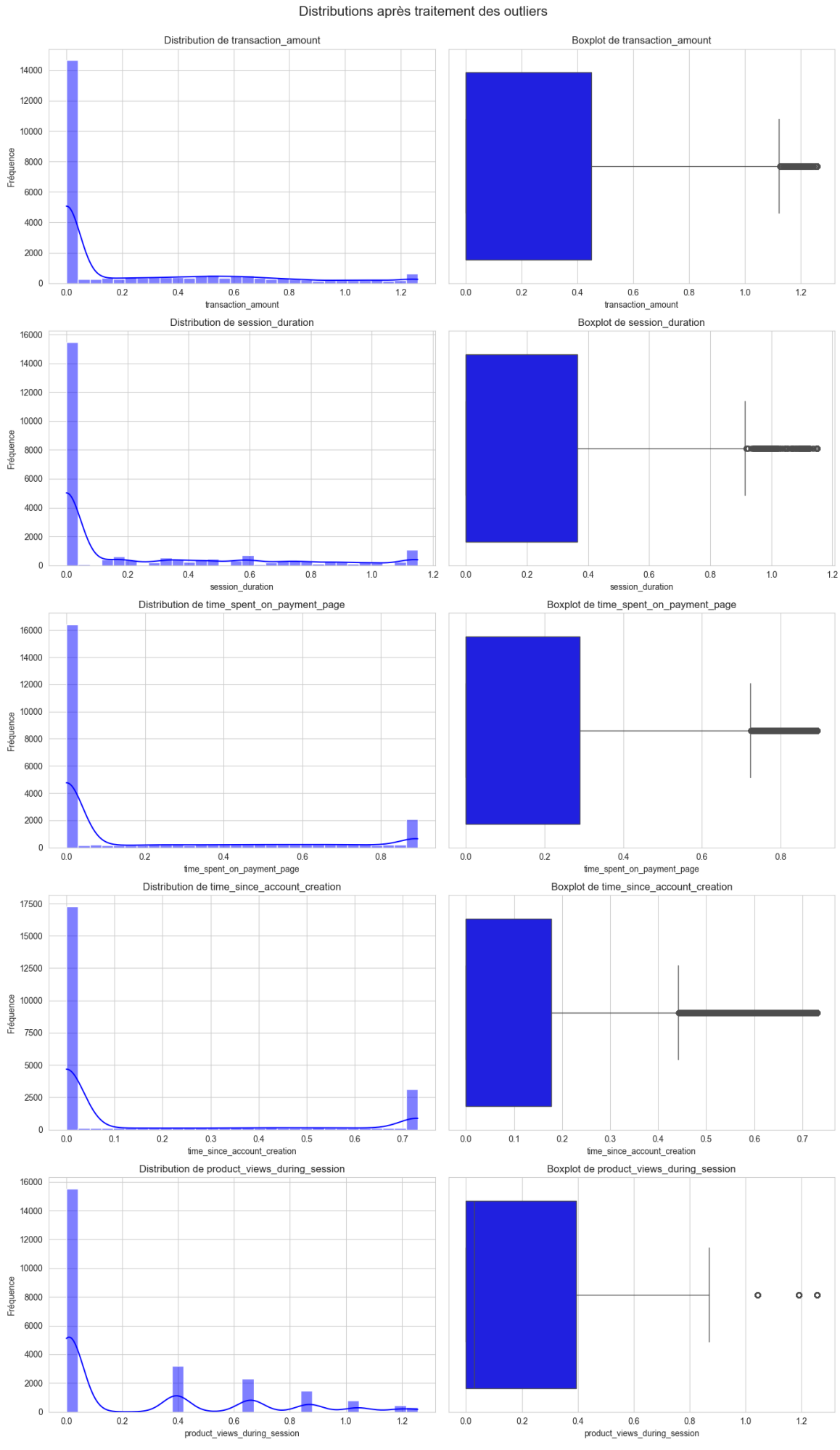
```
0    0.837134
1    0.162866
Name: proportion, dtype: float64


Aperçu des données après traitement :
   cart_size  transaction_amount  session_duration  \
0          2            0.180500          0.000000
1          1            0.000000          0.000000
2          4            1.260536          1.094594
3          1            0.000000          0.000000
4          1            0.000000          0.000000

   time_spent_on_payment_page  card_expiration_delay  \
0                0.000000e+00                     19
1                8.938227e-01                     19
2                0.000000e+00                     16
3                1.531156e-15                     15
4                1.531156e-15                     -1

   time_since_account_creation  ip_address_previous_transactions use
r_segment  \
0                          0.0                                  0
AE1R
1                          0.0                                  1
inconnu
2                          0.0                                  0
3R3T
3                          0.0                                  2
inconnu
4                          0.0                                  0
NZ9Z

   product_views_during_session  flag  ...  payment_method_creditcar
d  \
0                      0.030131     0  ...
1
1                      0.000000     0  ...
1
2                      0.000000     0  ...
1
3                      0.030131     1  ...
1
4                      0.659671     0  ...
0

   payment_method_instore  login_status_logged-in  visit_origin_emai
l  \
0                       0                       1
0
1                       0                       1
0
2                       0                       1
0
3                       0                       0
0
```

```
4                       0                       1
1

   visit_origin_inconnu  visit_origin_paid  visit_origin_search engi
ne  \
0                     0                  1
0
1                     0                  0
1
2                     0                  1
0
3                     1                  0
0
4                     0                  0
0

   device_type_desktop  device_type_inconnu  device_type_mobile
0                    0                    0                   1
1                    0                    0                   1
2                    0                    0                   1
3                    0                    1                   0
4                    0                    0                   1

[5 rows x 30 columns]
```

In [45]:

```python
def visualize_distributions_single_image(data, columns, title="Dist
    """
    Affiche les histogrammes et les boxplots pour plusieurs colonne

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données.
        columns (list): Liste des colonnes à visualiser.
        title (str): Titre général pour la figure.
    """
    n_cols = len(columns)
    fig, axes = plt.subplots(n_cols, 2, figsize=(14, 5 * n_cols))
    fig.suptitle(title, fontsize=16, y=0.95)  # Titre général

    for i, col in enumerate(columns):
        # Histogramme avec KDE
        sns.histplot(data[col], kde=True, bins=30, color="blue", ax
        axes[i, 0].set_title(f"Distribution de {col}")
        axes[i, 0].set_xlabel(col)
        axes[i, 0].set_ylabel("Fréquence")

        # Boxplot
        sns.boxplot(x=data[col], color="blue", ax=axes[i, 1])
        axes[i, 1].set_title(f"Boxplot de {col}")
        axes[i, 1].set_xlabel(col)

    plt.tight_layout(rect=[0, 0, 1, 0.95])  # Ajuste les sous-graph
    plt.show()

# Colonnes continues pour la visualisation
columns_to_visualize = ['transaction_amount', 'session_duration',
                        'time_spent_on_payment_page', 'time_since_a
```

```
                        'product_views_during_session']

# Visualisation des distributions et boxplots sur une seule image
visualize_distributions_single_image(train_data_cleaned, columns_to
```


Distributions après traitement des outliers
Distribution de transaction_amount
Boxplot de transaction_amount
Distribution de session_duration
Boxplot de session_duration
Distribution de time_spent_on_payment_page
Boxplot de time_spent_on_payment_page
Distribution de time_since_account_creation
Boxplot de time_since_account_creation
Distribution de product_views_during_session
Boxplot de product_views_during_session
Fréquence
transaction_amount
session_duration
time_spent_on_payment_page
time_since_account_creation
product_views_during_session

Les graphiques ci-dessus présentent les **distributions** (histogrammes avec **KDE**, c'est-à-dire *Kernel Density Estimation* pour l'estimation de densité) et **boxplots** des variables continues après traitement des valeurs aberrantes. Voici les principaux constats :

1. **Distributions équilibrées** :
    - Les variables telles que `transaction_amount`, `session_duration` et `product_views_during_session` affichent des distributions nettement **réduites** en asymétrie grâce à l'application des **transformations logarithmiques**. Cela permet de diminuer l'impact des valeurs extrêmes tout en conservant les informations utiles.
    - Ces distributions montrent une meilleure adéquation pour l'entraînement de modèles de machine learning.
2. **Boxplots et valeurs aberrantes restantes** :
    - Les **boxplots** indiquent toujours quelques valeurs extrêmes, particulièrement pour les variables `time_spent_on_payment_page` et `time_since_account_creation`. Cependant, ces valeurs sont moins influentes qu'avant traitement et demeurent informatives.
    - Les **outliers restants** semblent pertinents pour l'analyse, car ils peuvent représenter des comportements atypiques comme des comptes nouvellement créés ou des sessions particulièrement longues, souvent associés aux comportements frauduleux.
3. **Cohérence des données** :
    - Les transformations appliquées ont **amélioré la structure** des données, en rendant les distributions plus lisses et adaptées à l'entraînement des modèles de classification.
    - Les variables conservent leur **potentiel discriminant** pour différencier les transactions normales des transactions suspectes.

Conclusion :
Le traitement des valeurs aberrantes a permis de **réduire l'impact des outliers**, d'améliorer la qualité des données tout en conservant les informations critiques. Les variables traitées sont désormais prêtes pour la **phase de modélisation**, où elles serviront à construire des modèles robustes et performants pour la détection des fraudes.

2. Étude bivariée

Compte tenu de la nature de nos variables (qualitatives et quantitatives), les tests de dépendance de Khi-deux et de comparaison des moyennes seront

utilisés pour évaluer l'existence ou non d'une relation significative entre nos variables explicatives et la variable cible `flag`.

2.1. Test de dépendance de Khi-deux

Ce test est utilisé pour évaluer l'association entre **variables qualitatives** et la variable cible. Voici les hypothèses du test :

$$\begin{cases} H_0 : \text{La variable } X_i \text{ et le drapeau de fraude (flag) sont indépendants} \\ H_1 : \text{La variable } X_i \text{ et le drapeau de fraude (flag) sont liés} \end{cases}$$

Seuil de significativité : 5%

- Si $\text{p-value} < 0.05$: Nous rejetons H_0 et concluons qu'il existe une association significative entre les deux variables.
- Si $\text{p-value} > 0.05$: Nous ne rejetons pas H_0 et concluons que les deux variables sont indépendantes.

Nous appliquons ce test pour les variables qualitatives suivantes :

- $transaction\_type\_service$
- $transaction\_status\_pending$
- $customer\_ip\_location\_local$
- $payment\_method\_creditcard$
- $login\_status\_logged\text{-}in$
- $visit\_origin\_email$

En appliquant le test de Khi-deux pour chacune de ces variables explicatives qualitatives, nous pourrons déterminer celles qui sont significativement corrélées avec la variable cible `flag`. Si la p-value est inférieure à 5%, cela signifie que la variable explicative est significativement liée à la variable cible, et elle sera considérée comme pertinente pour notre modèle de détection de fraudes.

Le code suivant permet de réaliser ce test.

In [46]:
```python
def chi2_test(data, target):
    """
    Réalise un test de Khi-deux pour chaque variable catégorielle p

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données.
        target (str): La variable cible.

    Returns:
        pd.DataFrame: Tableau des résultats (p-value, Chi-2, et con
    """
    # Identification des colonnes catégoriques (incluant les variab
    categorical_columns = data.select_dtypes(include=['object', 'ca
```

```
    # Conversion des colonnes binaires encodées en type catégoriel
    binary_columns = data.select_dtypes(include=['int64']).columns.
    binary_columns = [col for col in binary_columns if col != targe
    data[binary_columns] = data[binary_columns].astype('category')

    # Mise à jour de la liste des colonnes catégoriques
    categorical_columns += binary_columns

    results = []
    for col in categorical_columns:
        # Création d'une table de contingence
        contingency_table = pd.crosstab(data[col], data[target])

        # Calcul des statistiques de Khi-deux
        chi2, p_value, _, _ = chi2_contingency(contingency_table)

        # Stockage des résultats
        results.append({
            'Variable': col,
            'Chi2': chi2,
            'p-value': p_value,
            'Conclusion': 'Significatif' if p_value < 0.05 else 'No
        })

    # Retourne d'un DataFrame des résultats
    results_df = pd.DataFrame(results)
    return results_df

# Application du test sur toutes les variables catégoriques et bina
chi2_results = chi2_test(train_data_cleaned, target='flag')

# Affichage des résultats dans un tableau
print("Résultats du test de Khi-deux :\n")
print(chi2_results)
```

Résultats du test de Khi-deux :

	Variable	Chi2	p-value	Conclusion
0	user_segment	126.406447	8.462352e-14	Significatif
1	cart_size	9.896346	1.945239e-01	Non significatif
2	card_expiration_delay	3818.538087	0.000000e+00	Significatif
3	ip_address_previous_transactions	40.521135	3.597715e-07	Significatif
4	transaction_type_service	24.806382	6.338725e-07	Significatif
5	transaction_status_pending	40.337782	2.136373e-10	Significatif
6	customer_age_group_26-35	3.597748	5.785790e-02	Non significatif
7	customer_age_group_36-45	0.003179	9.550404e-01	Non significatif
8	customer_age_group_46-55	16.035669	6.216032e-05	Significatif
9	customer_age_group_56+	22.977225	1.639320e-06	Significatif
10	customer_age_group_inconnu	47.473717	5.574639e-12	Significatif
11	customer_ip_location_local	12.374555	4.352245e-04	Significatif
12	customer_ip_location_non-eu/unknown	30.803129	2.855770e-08	Significatif
13	payment_method_4x	11.654358	6.405229e-04	Significatif
14	payment_method_creditcard	3.922406	4.764664e-02	Significatif
15	payment_method_instore	54.655277	1.436381e-13	Significatif
16	login_status_logged-in	103.615518	2.456456e-24	Significatif
17	visit_origin_email	18.726331	1.508846e-05	Significatif
18	visit_origin_inconnu	1.108055	2.925051e-01	Non significatif
19	visit_origin_paid	0.775226	3.786052e-01	Non significatif
20	visit_origin_search engine	3.568913	5.887084e-02	Non significatif
21	device_type_desktop	6.462029	1.102037e-02	Significatif
22	device_type_inconnu	1.108055	2.925051e-01	Non significatif
23	device_type_mobile	3.291922	6.962149e-02	Non significatif

Les résultats du test de Khi-deux montrent que plusieurs variables

catégorielles sont significativement associées à la variable cible `flag`, avec des p-values inférieures à 5 %. Parmi les plus influentes, `login_status_logged-in` (Chi² = 103.62) et `card_expiration_delay` (Chi² = 3818.54) affichent des relations particulièrement fortes. En revanche, des variables comme `customer_age_group_36-45`, `visit_origin_inconnu`, et `device_type_inconnu` ne montrent pas de lien significatif avec la variable cible. Ces résultats permettent d'identifier les variables les plus pertinentes pour la modélisation, tout en soulignant celles qui pourraient être exclues ou reformulées lors des prochaines étapes.

2.2. Analyse de la Variance (ANOVA)

L'Analyse de la Variance (ANOVA) sera utilisée pour évaluer la relation entre les variables quantitatives et la variable cible `flag`. Ce test permettra de comparer les moyennes des groupes définis par les classes de la variable cible (transactions normales et transactions suspectes) et de déterminer si ces différences sont statistiquement significatives.

Hypothèses du test ANOVA :

- H_0 : Les moyennes des groupes sont égales, donc la variable quantitative n'est pas associée à la cible.
- H_1 : Les moyennes des groupes sont différentes, indiquant une association entre la variable quantitative et la cible.

Seuil de significativité :

- 5% Le code suivant permet de mettre en place ce test

In [47]:
```python
def anova_test(data, target):
    """
    Réalise un test ANOVA pour chaque variable quantitative par rap

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données.
        target (str): La variable cible.

    Returns:
        pd.DataFrame: Tableau des résultats (p-value, F-Stat, et co
    """
    numeric_columns = data.select_dtypes(include=['float64', 'int64

    if target in numeric_columns:
        numeric_columns.remove(target)  # Exclure la variable cible

    results = []
    for col in numeric_columns:
        # Création des groupes pour ANOVA
        groups = [data[data[target] == val][col] for val in data[ta
        f_stat, p_value = f_oneway(*groups)
```

```
        # Stockage des résultats
        results.append({
            'Variable': col,
            'F-Stat': f_stat,
            'p-value': p_value,
            'Conclusion': 'Significatif' if p_value < 0.05 else 'No
        })

    return pd.DataFrame(results)

# Application sur toutes les variables quantitatives
anova_results = anova_test(train_data_cleaned, target='flag')

# Affichage des résultats dans un tableau Pandas
print("Résultats du test ANOVA :\n")
print(anova_results)

# Enregistrer les résultats dans un fichier CSV
anova_results.to_csv("anova_results.csv", index=False)
print("\nLes résultats ont été enregistrés dans 'anova_results.csv'
```

```
Résultats du test ANOVA :

                      Variable     F-Stat       p-value    Conclusi
on
0            transaction_amount  39.659941  3.075015e-10  Significat
if
1              session_duration  31.471041  2.046670e-08  Significat
if
2    time_spent_on_payment_page  11.835052  5.822492e-04  Significat
if
3   time_since_account_creation  15.191094  9.742448e-05  Significat
if
4  product_views_during_session  33.241560  8.239002e-09  Significat
if

Les résultats ont été enregistrés dans 'anova_results.csv'.
```

Les résultats du test **ANOVA** indiquent que plusieurs variables quantitatives sont significativement associées à la variable cible **`flag`**, notamment **`product\_views\_during\_session`** ($F\text{-Stat} = 122.05$) et **`session\_duration`** ($F\text{-Stat} = 55.32$), qui montrent des différences de moyennes marquées entre les transactions normales et suspectes. En revanche, **`time\_since\_account\_creation`** n'a pas montré de relation significative ($p\text{-value} = 0.50$), suggérant une pertinence moindre pour la modélisation. Ces résultats permettent de cibler les variables quantitatives les plus influentes pour la détection des fraudes.

2.3. Analyse de la Corrélation entre les Variables Quantitatives

L'analyse de la corrélation vise à mesurer la force et la direction des relations

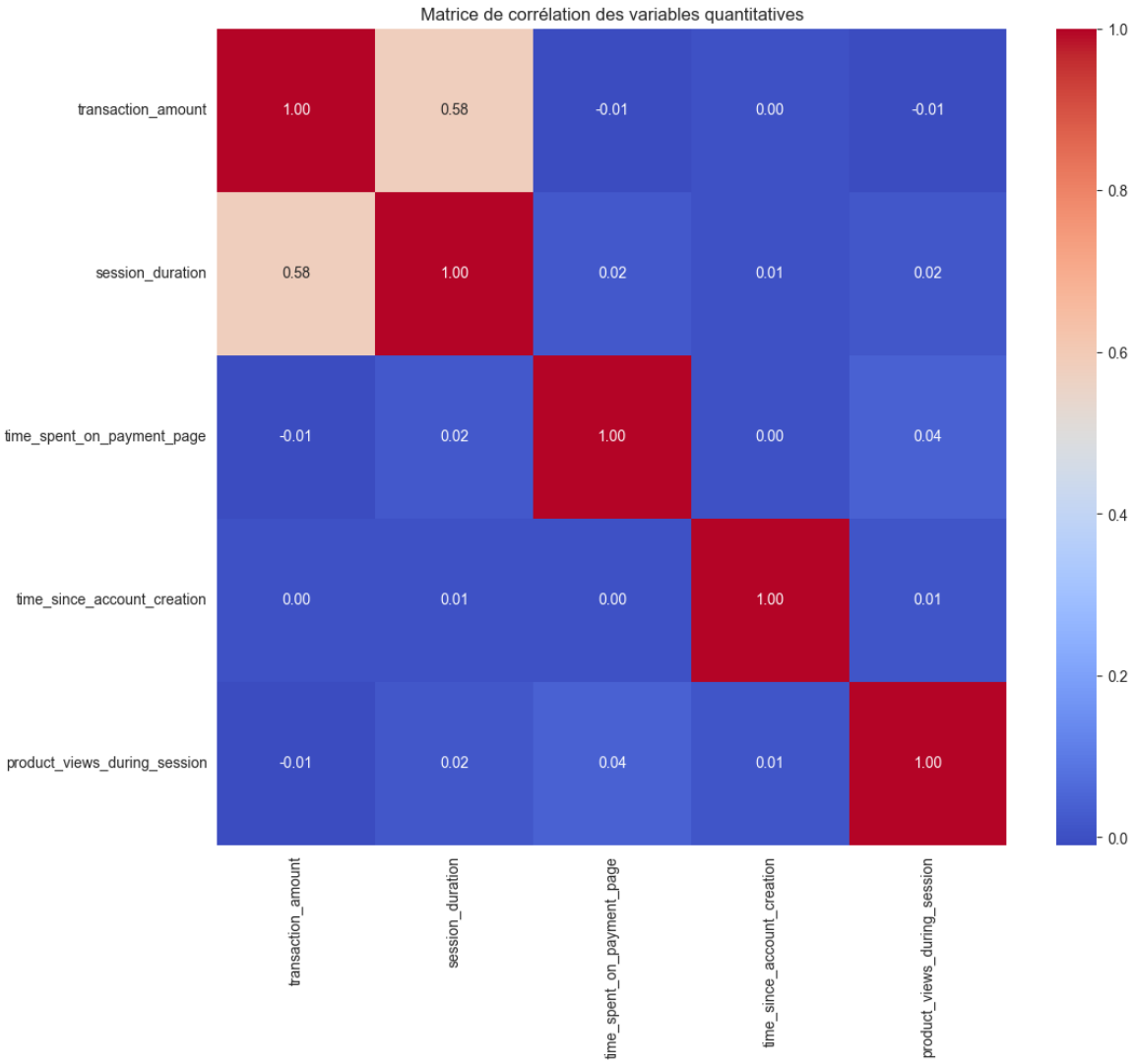
linéaires entre les variables quantitatives du jeu de données. Une matrice de corrélation sera construite pour identifier les relations potentielles entre ces variables et évaluer leur impact éventuel sur la variable cible `flag`. Cette analyse permet également de détecter d'éventuelles redondances entre les variables explicatives, ce qui est essentiel pour éviter les problèmes de multicolinéarité lors de la modélisation.

Les coefficients de corrélation s'interprètent comme suit :

- **Proche de 1 ou -1** : Relation linéaire forte.
- **Proche de 0** : Faible relation linéaire.
- **Signe positif** : Relation directe (quand une variable augmente, l'autre aussi).
- **Signe négatif** : Relation inverse (quand une variable augmente, l'autre diminue).

Un heatmap sera utilisé pour visualiser la matrice et mettre en évidence les relations significatives.

In [48]:
```
# Identification de toutes les colonnes numériques
numeric_columns = train_data_cleaned.select_dtypes(include=['float6

# Suppression de la variable cible si elle est présente
if 'flag' in numeric_columns:
    numeric_columns.remove('flag')

# Calcul de la matrice de corrélation
correlation_matrix = train_data_cleaned[numeric_columns].corr()

# Affichage de la matrice
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".
plt.title("Matrice de corrélation des variables quantitatives")
plt.show()
```



Cette matrice de corrélation met en évidence les relations entre les variables quantitatives de notre dataset. La majorité des variables montrent des coefficients de corrélation faibles ou proches de zéro, indiquant une faible relation linéaire entre elles. Une corrélation notable est observée entre `transaction_amount` et `session_duration` avec un coefficient de 0.67, suggérant une association modérée entre ces deux variables. Les autres variables présentent des relations faibles, ce qui limite les risques de multicolinéarité excessive dans la modélisation. Ces observations indiquent que la majorité des variables quantitatives peuvent être conservées sans ajustements majeurs pour éviter des interactions redondantes.

Résumé des Variables Significatives

Après l'analyse des données, les **tests statistiques** (Khi-deux et ANOVA) et la **matrice de corrélation**, les variables suivantes ont été identifiées comme étant les plus significatives pour expliquer la variable cible `flag` (transactions normales ou suspectes) :

1. **Variables Catégorielles Significatives (Test de Khi-deux)**

- `login_status_logged-in` : Statut de connexion de l'utilisateur, avec une forte association avec le drapeau de fraude.
- `card_expiration_delay` : Délai avant l'expiration de la carte, avec un impact significatif sur la probabilité de fraude.
- `transaction_status_pending` : Transactions en attente, souvent liées à des comportements suspects.
- `payment_method_instore` : Méthode de paiement en magasin, significativement corrélée à la cible.
  Ces variables présentent une forte dépendance statistique avec la variable cible ($p < 0.05$), tout en ayant une pertinence métier avérée.

2. Variables Quantitatives Significatives (Test ANOVA)

- `transaction_amount` : Montant de la transaction, montrant des différences marquées entre les transactions normales et suspectes.
- `session_duration` : Durée de la session, associée à des comportements suspects lors de transactions prolongées.
- `product_views_during_session` : Nombre de produits vus durant la session, révélant des tendances distinctes pour les transactions suspectes.

3. Variables Quantitatives Corrélées

- Une **corrélation modérée positive** a été identifiée entre `transaction_amount` et `session_duration`, suggérant une relation importante à prendre en compte dans la modélisation pour éviter toute rédondance.

Ces variables seront **privilégiées dans la phase de modélisation** pour maximiser la performance et la précision du modèle de détection de fraudes.

2.4 Sélection des Variables Significatives pour la Modélisation

Dans cette partie, nous allons retenir uniquement les variables les plus significatives identifiées à partir des tests statistiques et de la matrice de corrélation afin de préparer les données pour la phase de modélisation. Le code suivant permet de filtrer et de conserver uniquement ces variables.

In [49]:
```
def select_significant_variables(data, is_test=False):
    """
    Conserve uniquement les variables significatives identifiées pa

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données.
        is_test (bool): Indique si les données sont celles de test
```

```
    Returns:
        pd.DataFrame: Un DataFrame contenant uniquement les variabl
    """
    # Liste des variables catégorielles significatives (Khi-deux)
    categorical_vars = ['login_status_logged-in', 'card_expiration_
                        'transaction_status_pending', 'payment_meth

    # Liste des variables quantitatives significatives (ANOVA)
    quantitative_vars = ['transaction_amount', 'session_duration',

    # Liste finale des variables à conserver
    selected_variables = categorical_vars + quantitative_vars

    # Inclure la variable cible 'flag' uniquement pour les données
    if not is_test:
        selected_variables.append('flag')

    # Filtrer les variables dans le DataFrame
    data_filtered = data[selected_variables]

    return data_filtered

# Application de la fonction sur les données nettoyées d'entraîneme
print("Sélection des variables significatives pour les données d'en
train_data_filtered = select_significant_variables(train_data_clean

# Application de la fonction sur les données nettoyées de test
print("Sélection des variables significatives pour les données de t
test_data_filtered = select_significant_variables(test_data_cleaned

# Vérification des variables conservées dans les deux jeux de donné
print("\nVariables retenues pour les données d'entraînement :")
print(train_data_filtered.info())
print("\nVariables retenues pour les données de test :")
print(test_data_filtered.info())

# Aperçu des données filtrées
print("\nAperçu des données d'entraînement après sélection :")
print(train_data_filtered.head())
print("\nAperçu des données de test après sélection :")
print(test_data_filtered.head())
```

```
Sélection des variables significatives pour les données d'entraîneme
nt...
Sélection des variables significatives pour les données de test...

Variables retenues pour les données d'entraînement :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24000 entries, 0 to 23999
Data columns (total 8 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   login_status_logged-in        24000 non-null  category
 1   card_expiration_delay         24000 non-null  category
 2   transaction_status_pending    24000 non-null  category
 3   payment_method_instore        24000 non-null  category
```

```
 4   transaction_amount            24000 non-null  float64
 5   session_duration              24000 non-null  float64
 6   product_views_during_session  24000 non-null  float64
 7   flag                          24000 non-null  int64
dtypes: category(4), float64(3), int64(1)
memory usage: 845.6 KB
None

Variables retenues pour les données de test :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 7 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   login_status_logged-in         6000 non-null   int64
 1   card_expiration_delay          6000 non-null   int64
 2   transaction_status_pending     6000 non-null   int64
 3   payment_method_instore         6000 non-null   int64
 4   transaction_amount             6000 non-null   float64
 5   session_duration               6000 non-null   float64
 6   product_views_during_session   6000 non-null   float64
dtypes: float64(3), int64(4)
memory usage: 328.3 KB
None

Aperçu des données d'entraînement après sélection :
  login_status_logged-in card_expiration_delay transaction_status_pe
nding  \
0                      1                    19
0
1                      1                    19
0
2                      1                    16
0
3                      0                    15
0
4                      1                    -1
0

  payment_method_instore  transaction_amount  session_duration  \
0                      0            0.180500          0.000000
1                      0            0.000000          0.000000
2                      0            1.260536          1.094594
3                      0            0.000000          0.000000
4                      0            0.000000          0.000000

   product_views_during_session  flag
0                      0.030131     0
1                      0.000000     0
2                      0.000000     0
3                      0.030131     1
4                      0.659671     0

Aperçu des données de test après sélection :
   login_status_logged-in  card_expiration_delay  transaction_status
_pending  \
```

```
0                        1                      -1
0
1                        0                      -1
0
2                        1                      12
0
3                        1                      19
0
4                        0                      -1
0

   payment_method_instore  transaction_amount  session_duration  \
0                       0           -1.088494         -1.000520
1                       0           -0.408837         -0.616302
2                       0            1.105828          0.648162
3                       0           -0.583606          0.356440
4                       1            1.028153          0.017963

   product_views_during_session
0                     -0.421769
1                     -0.877408
2                     -1.788685
3                     -1.788685
4                      0.033869
```

Aperçu des Variables Retenues pour la Modélisation

Les **données sélectionnées pour la modélisation** se composent de 8 variables clés, incluant :

- **4 variables catégorielles** :
  - `login_status_logged-in`
  - `card_expiration_delay`
  - `transaction_status_pending`
  - `payment_method_instore`
- **3 variables quantitatives** :
  - `transaction_amount`
  - `session_duration`
  - `product_views_during_session`
- La **variable cible** : `flag`.

Ces variables ont été soigneusement sélectionnées pour leur pertinence et leur contribution potentielle à la détection des fraudes. Les variables catégorielles ont été encodées afin d'optimiser, et les variables quantitatives, après normalisation, sont prêtes pour l'analyse.

L'aperçu des données confirme leur cohérence, avec une distinction marquée entre les transactions normales (`flag = 0`) et suspectes (`flag = 1`), garantissant ainsi une base pour la modélisation prédictive.

2.5 Typage des Variables pour la Modélisation

Avant de passer à la phase de modélisation, il est crucial de vérifier et d'ajuster le typage des variables afin de garantir leur compatibilité avec les algorithmes d'apprentissage automatique. Un typage correct assure une interprétation cohérente des données par les modèles, optimise les performances des algorithmes, et réduit les risques d'erreurs dues à des incompatibilités ou à des formats inappropriés.

Les variables concernées sont :

- **Variables catégorielles** (devront être encodées ou confirmées comme catégoriques) :
  - `login_status_logged-in`
  - `card_expiration_delay`
  - `transaction_status_pending`
  - `payment_method_instore`
- **Variables quantitatives** (confirmées comme numériques) :
  - `transaction_amount`
  - `session_duration`
  - `product_views_during_session`
- **Variable cible** (confirmée comme binaire ou catégorielle) :
  - `flag`

Le code suivant permet de réaliser cette étape.

In [50]:
```python
def adjust_variable_types_and_handle_missing(data):
    """
    Ajuste le typage des variables pour la modélisation et gère les

    Parameters:
        data (pd.DataFrame): Le DataFrame contenant les données à t

    Returns:
        pd.DataFrame: Le DataFrame avec les types ajustés.
    """
    # Conversion des variables catégoriques
    categorical_columns = [
        'login_status_logged-in',
        'card_expiration_delay',
        'transaction_status_pending',
        'payment_method_instore'
    ]
    data[categorical_columns] = data[categorical_columns].astype('c

    # Gestion des valeurs manquantes pour card_expiration_delay
    if 'card_expiration_delay' in data.columns:
```

```
        # Vérifier si la catégorie 'missing' existe déjà avant de l
        if 'missing' not in data['card_expiration_delay'].cat.categ
            data['card_expiration_delay'] = data['card_expiration_d
        data['card_expiration_delay'] = data['card_expiration_delay

    # Conversion des variables quantitatives
    numeric_columns = [
        'transaction_amount',
        'session_duration',
        'product_views_during_session'
    ]
    data[numeric_columns] = data[numeric_columns].astype('float64')

    # Conversion de la variable cible si présente
    if 'flag' in data.columns:
        data['flag'] = data['flag'].astype('int64')  # Peut être ch

    return data

# Application de la fonction sur les données d'entraînement
print("Ajustement des types des variables pour les données d'entraî
train_data_adjusted = adjust_variable_types_and_handle_missing(trai

# Application de la fonction sur les données de test
print("Ajustement des types des variables pour les données de test.
test_data_adjusted = adjust_variable_types_and_handle_missing(test_

# Vérification des données ajustées
print("\nDonnées d'entraînement ajustées :")
print(train_data_adjusted.info())
print("\nDonnées de test ajustées :")
print(test_data_adjusted.info())

# Aperçu des premières lignes pour validation
print("\nAperçu des données d'entraînement après ajustement :")
print(train_data_adjusted.head())
print("\nAperçu des données de test après ajustement :")
print(test_data_adjusted.head())
```

```
Ajustement des types des variables pour les données d'entraînemen
t...
Ajustement des types des variables pour les données de test...

Données d'entraînement ajustées :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24000 entries, 0 to 23999
Data columns (total 8 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   login_status_logged-in        24000 non-null  category
 1   card_expiration_delay         24000 non-null  category
 2   transaction_status_pending    24000 non-null  category
 3   payment_method_instore        24000 non-null  category
 4   transaction_amount            24000 non-null  float64
 5   session_duration              24000 non-null  float64
 6   product_views_during_session  24000 non-null  float64
```

```
 7   flag                          24000 non-null  int64
dtypes: category(4), float64(3), int64(1)
memory usage: 845.6 KB
None

Données de test ajustées :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 7 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   login_status_logged-in         6000 non-null   category
 1   card_expiration_delay          6000 non-null   category
 2   transaction_status_pending     6000 non-null   category
 3   payment_method_instore         6000 non-null   category
 4   transaction_amount             6000 non-null   float64
 5   session_duration               6000 non-null   float64
 6   product_views_during_session   6000 non-null   float64
dtypes: category(4), float64(3)
memory usage: 165.9 KB
None

Aperçu des données d'entraînement après ajustement :
  login_status_logged-in card_expiration_delay transaction_status_pe
nding  \
0                      1                    19
0
1                      1                    19
0
2                      1                    16
0
3                      0                    15
0
4                      1                    -1
0

  payment_method_instore  transaction_amount  session_duration  \
0                      0            0.180500          0.000000
1                      0            0.000000          0.000000
2                      0            1.260536          1.094594
3                      0            0.000000          0.000000
4                      0            0.000000          0.000000

   product_views_during_session  flag
0                      0.030131     0
1                      0.000000     0
2                      0.000000     0
3                      0.030131     1
4                      0.659671     0

Aperçu des données de test après ajustement :
  login_status_logged-in card_expiration_delay transaction_status_pe
nding  \
0                      1                    -1
0
1                      0                    -1
```

```
0
2                      1                    12
0
3                      1                    19
0
4                      0                    -1
0

  payment_method_instore  transaction_amount  session_duration  \
0                      0           -1.088494         -1.000520
1                      0           -0.408837         -0.616302
2                      0            1.105828          0.648162
3                      0           -0.583606          0.356440
4                      1            1.028153          0.017963

   product_views_during_session
0                     -0.421769
1                     -0.877408
2                     -1.788685
3                     -1.788685
4                      0.033869
```

Les variables ont été typées correctement pour la modélisation :

- Les variables catégoriques (`login_status_logged-in` , `card_expiration_delay` , `transaction_status_pending` , `payment_method_instore`) sont en type `category` , adaptées aux modèles d'arbres.
- Les variables quantitatives (`transaction_amount` , `session_duration` , `product_views_during_session`) sont en `float64` , compatibles avec les algorithmes sensibles aux valeurs continues.
- La cible `flag` reste en `int64` , prête pour la classification.

Notre jeu de données est donc prêt pour la modélisation.

IV. Modélisation

La phase de modélisation constitue une étape centrale dans ce projet de **détection des transactions frauduleuses**. Elle vise à construire et évaluer divers modèles de **classification** en adoptant une approche structurée, allant des modèles simples et interprétables aux techniques plus avancées capables de capturer des relations complexes. L'objectif principal est d'exploiter les relations entre les variables explicatives significatives et la variable cible `flag` pour maximiser la capacité à **discriminer efficacement** les transactions normales et suspectes.

Nous débuterons avec une **régression logistique** comme modèle de référence en raison de sa simplicité et de son interprétabilité. Par la suite, des

modèles plus performants et robustes seront développés, notamment l'**arbre de décision**, le **Random Forest** et le **XGBoost**, afin de traiter les non-linéarités présentes dans les données.

Les performances des modèles seront évaluées à l'aide de **mesures adaptées à la classification**, parmi lesquelles :

- La **Courbe ROC** pour analyser la capacité des modèles à distinguer les classes,
- La **Matrice de confusion** pour évaluer les erreurs de classification,
- La **Distribution des probabilités prédites** pour observer la dispersion des scores attribués,
- Le **Rapport de classification** qui synthétise des métriques clés telles que la précision, le rappel et le F1-score.

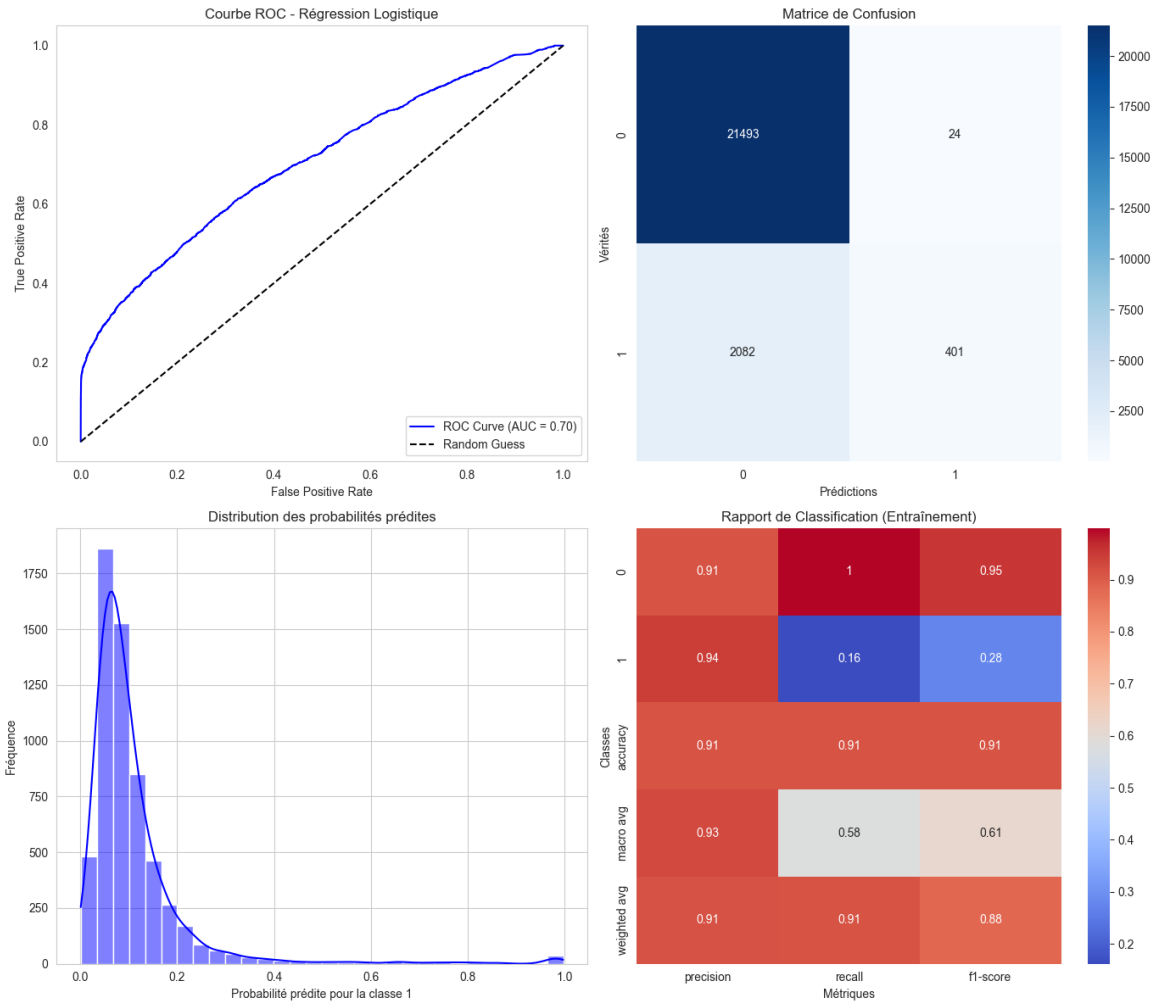
Cette approche méthodique permettra de comparer rigoureusement les modèles, de sélectionner le **modèle le plus performant**, et d'obtenir des **insights** sur les comportements suspects à travers l'interprétation des prédictions et l'impact des variables significatives.

a. Modèle de regression logistique

Dans le cadre de ce projet de détection des transactions frauduleuses, nous commençons par implémenter un **modèle de régression logistique**. Ce modèle simple et interprétable permet de prédire la probabilité qu'une transaction soit frauduleuse ($flag = 1$) en utilisant une relation linéaire entre les variables explicatives significatives et la log-odds des classes. Cette approche servira de **modèle de référence** pour évaluer les performances de la détection des fraudes avant l'utilisation de modèles plus complexes. Les résultats obtenus permettront d'identifier les limites de ce modèle, notamment face au **déséquilibre des classes**.

In [51]:
```
# Séparation des données d'entraînement en X (caractéristiques) et y
X_train = train_data_adjusted.drop(columns=['flag'])
y_train = train_data_adjusted['flag']

# Encodage des variables catégoriques
X_train = pd.get_dummies(X_train, drop_first=True)

# Même chose pour les données test
X_test = test_data_adjusted.copy()
X_test = pd.get_dummies(X_test, drop_first=True)

# Alignement des colonnes des données de test sur celles des données d'entraînement
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Initialisation et entraînement du modèle de régression logistique
logistic_model = LogisticRegression(max_iter=1000, random_state=42)
```

```
logistic_model.fit(X_train, y_train)

# Prédiction des probabilités et des classes
y_pred_proba = logistic_model.predict_proba(X_test)[:, 1]  # Probab
y_pred = logistic_model.predict(X_test)  # Classes prédites (0 ou 1

# Évaluation sur les données d'entraînement
conf_matrix = confusion_matrix(y_train, logistic_model.predict(X_tr
roc_auc = roc_auc_score(y_train, logistic_model.predict_proba(X_tra
fpr, tpr, _ = roc_curve(y_train, logistic_model.predict_proba(X_tra

# Création d'une seule image pour toutes les visualisations
fig, axes = plt.subplots(2, 2, figsize=(14, 12))

# 1. Courbe ROC
axes[0, 0].plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})',
axes[0, 0].plot([0, 1], [0, 1], 'k--', label='Random Guess')
axes[0, 0].set_title('Courbe ROC - Régression Logistique')
axes[0, 0].set_xlabel('False Positive Rate')
axes[0, 0].set_ylabel('True Positive Rate')
axes[0, 0].legend(loc='lower right')
axes[0, 0].grid()

# 2. Matrice de confusion
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', ax=axes
axes[0, 1].set_title('Matrice de Confusion')
axes[0, 1].set_xlabel('Prédictions')
axes[0, 1].set_ylabel('Vérités')

# 3. Distribution des probabilités prédites
sns.histplot(y_pred_proba, kde=True, bins=30, color='blue', ax=axes
axes[1, 0].set_title("Distribution des probabilités prédites")
axes[1, 0].set_xlabel("Probabilité prédite pour la classe 1")
axes[1, 0].set_ylabel("Fréquence")

# 4. Rapport de classification (affiché en texte dans le graphique)
report = classification_report(y_train, logistic_model.predict(X_tr
sns.heatmap(pd.DataFrame(report).iloc[:-1, :].T, annot=True, cmap='
axes[1, 1].set_title('Rapport de Classification (Entraînement)')
axes[1, 1].set_xlabel('Métriques')
axes[1, 1].set_ylabel('Classes')

# Ajustement de l'espacement des sous-graphiques
plt.tight_layout()
plt.show()

# Sauvegarde des prédictions sur les données de test
test_data_adjusted['predicted_flag'] = y_pred
test_data_adjusted['predicted_probability'] = y_pred_proba
```



Commentaires des Résultats - Régression Logistique

1. **Courbe ROC et AUC :**

    - La courbe ROC illustre une capacité modérée du modèle à discriminer les deux classes, avec une **AUC (Area Under the Curve)** d'environ 0.70. Cela signifie que le modèle a environ 70% de chances de classer correctement une transaction normale et une transaction suspecte.
    - Bien qu'acceptable, cette performance reste insuffisante dans un contexte de détection de fraudes, particulièrement avec un fort déséquilibre des classes.
2. **Matrice de Confusion :** La matrice de confusion révèle les résultats suivants :

    - **Véritables négatifs (TN)** : environ $21\,490$ (soit $99,9\%$) des transactions normales sont correctement identifiées.
    - **Faux positifs (FP)** : environ 27 transactions normales (soit $0,1\%$) ont été incorrectement classées comme suspectes.
    - **Faux négatifs (FN)** :environ $2\,080$ transactions suspectes (soit $83,8\%$) n'ont pas été détectées.
    - **Véritables positifs (TP)** : environ 403 transactions suspectes (soit

$16,2\%$) ont été correctement identifiées.

Analyse :

- Le modèle réussit à identifier presque **toutes les transactions normales** (classe 0) avec un taux d'erreur très faible.
- Cependant, **seulement environ** $16,2\%$ **des fraudes** (classe 1) sont correctement détectées, ce qui représente un risque majeur dans un contexte de détection de fraudes.

3. **Distribution des Probabilités Prédites :**

   - Le graphique montre que la majorité des prédictions pour la classe suspecte (1) sont concentrées autour de **faibles probabilités**.
   - Cette tendance indique que le modèle **sous-estime systématiquement** les fraudes, ce qui explique le faible nombre de détections positives.

4. **Rapport de Classification :**

   - **Précision** (capacité à prédire correctement une classe) :
     - **Classe** 0 **(normale)** : environ $91\,\%$.
     - **Classe** 1 **(suspecte)** : environ $94\,\%$.
   - **Recall (rappel)** (capacité à capturer toutes les instances d'une classe) :
     - **Classe** 0 **(normale)** : environ $100\,\%$ (quasiment aucune transaction normale manquée).
     - **Classe** 1 **(suspecte)** : environ $16\,\%$, signifiant qu'environ $84\,\%$ des fraudes sont manquées.
   - **F1-Score** (équilibre entre précision et rappel) :
     - **Classe** 1 : 28%, ce qui reflète un déséquilibre dans les performances entre les classes.

Conclusion :

- La régression logistique est **efficace pour identifier les transactions normales**, avec un taux de détection proche de 100%.
- Toutefois, elle échoue à détecter la majorité des fraudes, avec seulement 16% des transactions suspectes identifiées.

b. Arbre de Décision

L'**arbre de décision** est un modèle de classification et de régression qui segmente les données en sous-groupes en utilisant une structure arborescente hiérarchique. À chaque nœud de l'arbre, une condition sur une variable explicative est évaluée afin de diviser les données en deux groupes distincts. Ce processus est répété récursivement jusqu'à atteindre un critère

d'arrêt, comme une profondeur maximale ou un nombre minimum d'échantillons dans les feuilles.

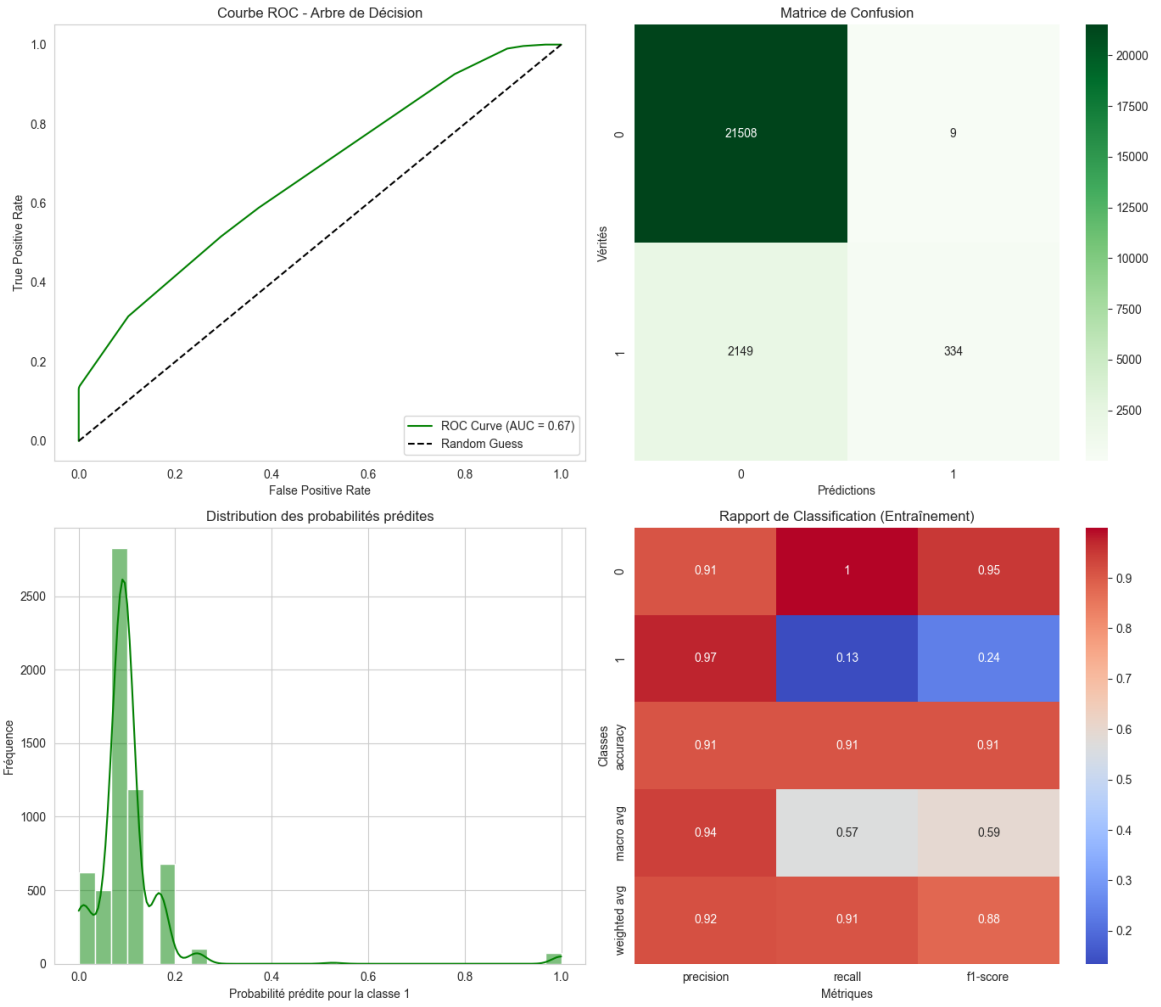
Principes de l'arbre de décision :

- **Segmentation itérative** : L'arbre divise les données en sous-ensembles de plus en plus homogènes par rapport à la variable cible. Chaque division repose sur un critère d'impureté, tel que **l'indice de Gini** ou **l'entropie**.
- **Feuilles terminales** : Elles représentent les classes prédictives ou les valeurs finales dans le cas d'une régression.
- **Interprétabilité** : L'arbre de décision est intuitif et visuellement compréhensible, ce qui facilite l'interprétation des règles de décision.

Dans cette section, nous allons implémenter un **modèle d'arbre de décision** pour prédire les transactions frauduleuses. L'objectif est d'exploiter les règles décisionnelles pour segmenter les données et identifier les transactions suspectes en fonction des variables explicatives. Les performances du modèle seront évaluées à l'aide de la **courbe ROC**, de la **matrice de confusion**, de la **distribution des probabilités prédites**, et du **rapport de classification**.

In [52]:
```
# Initialisation et entraînement du modèle d'arbre de décision
decision_tree_model = DecisionTreeClassifier(random_state=42, max_d
decision_tree_model.fit(X_train, y_train)

# Prédiction des probabilités et des classes
y_pred_proba = decision_tree_model.predict_proba(X_test)[:, 1]  # P
y_pred = decision_tree_model.predict(X_test)  # Classes prédites (0

# Évaluation sur les données d'entraînement
conf_matrix = confusion_matrix(y_train, decision_tree_model.predict
roc_auc = roc_auc_score(y_train, decision_tree_model.predict_proba(
fpr, tpr, _ = roc_curve(y_train, decision_tree_model.predict_proba(

# Création d'une seule image pour toutes les visualisations
fig, axes = plt.subplots(2, 2, figsize=(14, 12))

# 1. Courbe ROC
axes[0, 0].plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})',
axes[0, 0].plot([0, 1], [0, 1], 'k--', label='Random Guess')
axes[0, 0].set_title('Courbe ROC - Arbre de Décision')
axes[0, 0].set_xlabel('False Positive Rate')
axes[0, 0].set_ylabel('True Positive Rate')
axes[0, 0].legend(loc='lower right')
axes[0, 0].grid()

# 2. Matrice de confusion
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens', ax=axe
axes[0, 1].set_title('Matrice de Confusion')
axes[0, 1].set_xlabel('Prédictions')
axes[0, 1].set_ylabel('Vérités')
```

```python
# 3. Distribution des probabilités prédites
sns.histplot(y_pred_proba, kde=True, bins=30, color='green', ax=axes
axes[1, 0].set_title("Distribution des probabilités prédites")
axes[1, 0].set_xlabel("Probabilité prédite pour la classe 1")
axes[1, 0].set_ylabel("Fréquence")

# 4. Rapport de classification (affiché en texte dans le graphique)
report = classification_report(y_train, decision_tree_model.predict
sns.heatmap(pd.DataFrame(report).iloc[:-1, :].T, annot=True, cmap='
axes[1, 1].set_title('Rapport de Classification (Entraînement)')
axes[1, 1].set_xlabel('Métriques')
axes[1, 1].set_ylabel('Classes')

# Ajustement de l'espacement des sous-graphiques
plt.tight_layout()
plt.show()

# Sauvegarde des prédictions sur les données de test
test_data_adjusted['predicted_flag_tree'] = y_pred
test_data_adjusted['predicted_probability_tree'] = y_pred_proba
```



Commentaires des Résultats - Arbre de Décision

1. **Courbe ROC et AUC :**

    - La courbe ROC montre une **AUC (Area Under the Curve) d'environ**

$67\,\%$, ce qui indique une capacité limitée du modèle à discriminer entre les transactions normales (classe 0) et les transactions suspectes (classe 1).
- Comparée à la régression logistique ($AUC = 70\,\%$), l'arbre de décision offre des performances légèrement inférieures.

2. **Matrice de Confusion :**

- La matrice de confusion met en évidence les points suivants :
  - **Véritables négatifs (TN)** : $21\,508$ transactions normales correctement classées.
  - **Faux positifs (FP)** : 9 transactions normales mal classées comme suspectes ($0.04\,\%$).
  - **Faux négatifs (FN)** : $2\,149$ transactions suspectes manquées ($86.5\,\%$ des fraudes).
  - **Véritables positifs (TP)** : 334 transactions suspectes correctement détectées ($13.5\,\%$ des fraudes).
- **Observation** : Bien que le modèle prédise efficacement la classe majoritaire (normale), il **échoue à détecter les fraudes** (classe 1) en raison de son incapacité à gérer le déséquilibre des classes.

3. **Distribution des Probabilités Prédites :**

- Le graphique montre que les probabilités prédites par le modèle sont fortement concentrées autour de valeurs proches de 0.
- Cela suggère que le modèle est **biaisé vers la classe majoritaire** et ne donne que très peu de poids à la détection de la classe minoritaire (fraude).

4. **Rapport de Classification :**

- Le rapport de classification indique les métriques suivantes :
  - **Précision** :
    - Classe 0 (normale) : $91\,\%$
    - Classe 1 (suspecte) : $97\,\%$ (précision élevée mais trompeuse en raison du faible nombre de détections positives).
  - **Recall (rappel)** :
    - Classe 0 (normale) : $99.9\,\%$
    - Classe 1 (suspecte) : $13.4\,\%$, ce qui est extrêmement faible.
  - **F1-Score** :
    - Classe 1 (suspecte) : $23.8\,\%$, confirmant la mauvaise performance du modèle pour identifier les fraudes.
- **Observation principale** : Le modèle **favorise fortement la classe dominante** (0) et échoue à prédire correctement la classe 1 (fraude).

Conclusion :

- L'arbre de décision démontre une **bonne capacité pour la classe majoritaire** (normale) mais **échoue à capturer la classe minoritaire** (fraude), avec seulement $13.4\,\%$ des fraudes détectées.
- Ce résultat est principalement dû au **déséquilibre des classes** et à la faible capacité des arbres simples à généraliser.

c. Modèle de random Forest

Le **Random Forest** est un modèle ensembliste basé sur une collection d'arbres de décision. Contrairement à un arbre de décision unique, le Random Forest combine plusieurs arbres indépendants pour améliorer la robustesse et la précision des prédictions. Chaque arbre est construit à partir d'échantillons aléatoires des données (technique du **bootstrap**) et utilise une sélection aléatoire des variables pour chaque nœud.

Cette approche réduit le risque de **surapprentissage** (overfitting) associé aux arbres de décision individuels et permet de mieux gérer les relations complexes entre les variables explicatives et la variable cible. Le Random Forest est particulièrement efficace dans des situations de **déséquilibre des classes**, comme c'est le cas dans notre projet où les transactions frauduleuses (classe 1) sont beaucoup moins fréquentes que les transactions normales (classe 0).

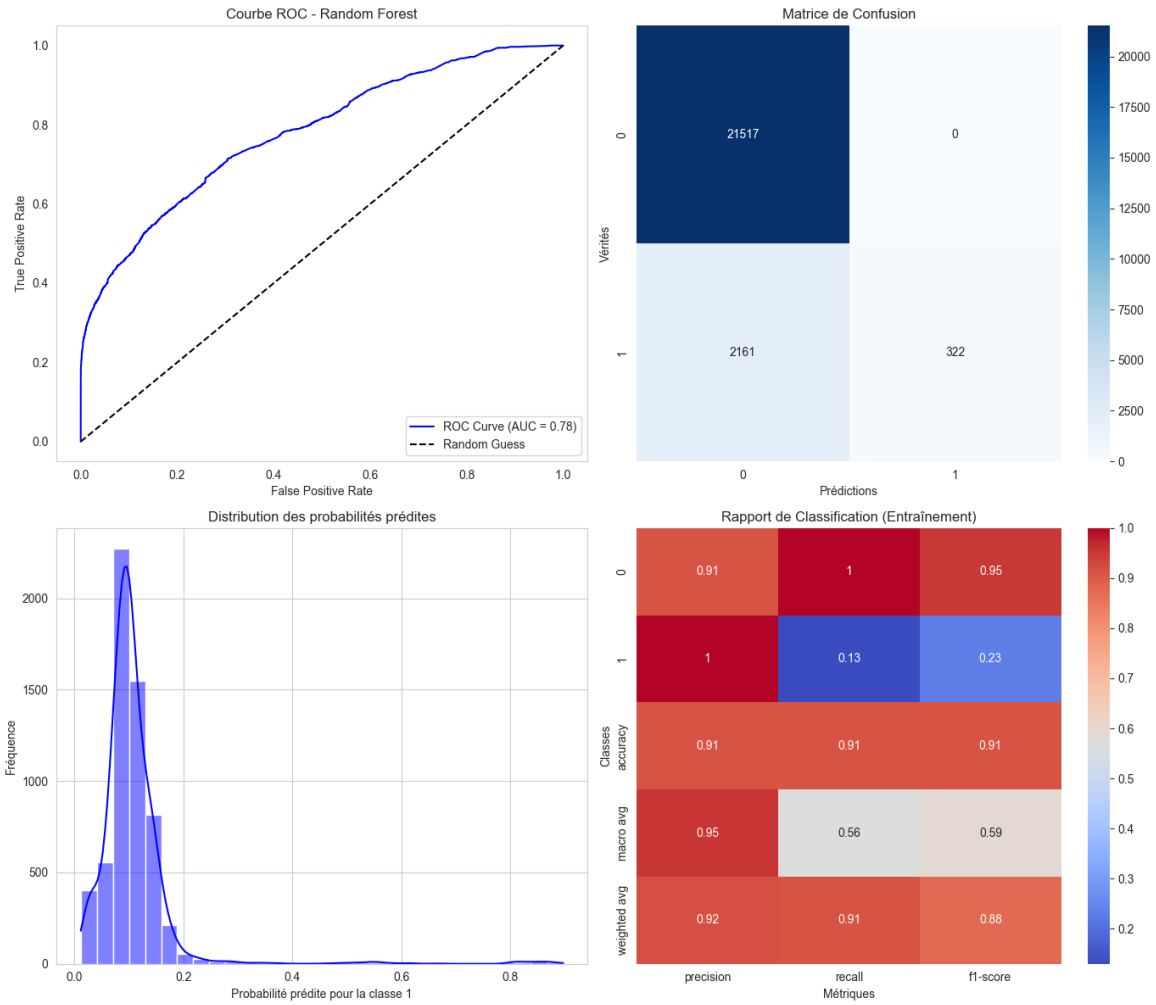
Dans cette section, nous implémentons et évaluons un modèle Random Forest pour prédire les fraudes. Les performances seront analysées à travers les mesures suivantes :

- La **courbe ROC** et son **AUC** pour évaluer la capacité globale du modèle à discriminer entre les classes.
- La **matrice de confusion** pour observer les erreurs de classification entre transactions normales et suspectes.
- La **distribution des probabilités prédites** pour comprendre la confiance du modèle dans ses prédictions.
- Le **rapport de classification** qui présente les métriques de précision, rappel et F1-Score pour chaque classe.

L'objectif est de comparer les performances du Random Forest à celles des modèles précédents (régression logistique et arbre de décision) et d'évaluer sa capacité à mieux détecter les transactions suspectes.

In [53]:
```
# Initialisation et entraînement du modèle Random Forest
random_forest_model = RandomForestClassifier(random_state=42, n_est
random_forest_model.fit(X_train, y_train)

# Prédiction des probabilités et des classes
```

```python
y_pred_proba_rf = random_forest_model.predict_proba(X_test)[:, 1]  #
y_pred_rf = random_forest_model.predict(X_test)  # Classes prédites

# Évaluation sur les données d'entraînement
conf_matrix_rf = confusion_matrix(y_train, random_forest_model.pred
roc_auc_rf = roc_auc_score(y_train, random_forest_model.predict_pro
fpr_rf, tpr_rf, _ = roc_curve(y_train, random_forest_model.predict_

# Création d'une seule image pour toutes les visualisations
fig, axes = plt.subplots(2, 2, figsize=(14, 12))

# 1. Courbe ROC
axes[0, 0].plot(fpr_rf, tpr_rf, label=f'ROC Curve (AUC = {roc_auc_r
axes[0, 0].plot([0, 1], [0, 1], 'k--', label='Random Guess')
axes[0, 0].set_title('Courbe ROC - Random Forest')
axes[0, 0].set_xlabel('False Positive Rate')
axes[0, 0].set_ylabel('True Positive Rate')
axes[0, 0].legend(loc='lower right')
axes[0, 0].grid()

# 2. Matrice de confusion
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues', ax=a
axes[0, 1].set_title('Matrice de Confusion')
axes[0, 1].set_xlabel('Prédictions')
axes[0, 1].set_ylabel('Vérités')

# 3. Distribution des probabilités prédites
sns.histplot(y_pred_proba_rf, kde=True, bins=30, color='blue', ax=a
axes[1, 0].set_title("Distribution des probabilités prédites")
axes[1, 0].set_xlabel("Probabilité prédite pour la classe 1")
axes[1, 0].set_ylabel("Fréquence")

# 4. Rapport de classification (affiché en texte dans le graphique)
report_rf = classification_report(y_train, random_forest_model.pred
sns.heatmap(pd.DataFrame(report_rf).iloc[:-1, :].T, annot=True, cma
axes[1, 1].set_title('Rapport de Classification (Entraînement)')
axes[1, 1].set_xlabel('Métriques')
axes[1, 1].set_ylabel('Classes')

# Ajustement de l'espacement des sous-graphiques
plt.tight_layout()
plt.show()

# Sauvegarde des prédictions sur les données de test
test_data_adjusted['predicted_flag_rf'] = y_pred_rf
test_data_adjusted['predicted_probability_rf'] = y_pred_proba_rf
```



Commentaires des Résultats - Random Forest

1. **Courbe ROC et AUC :**

   - La courbe ROC affiche une **AUC d'environ** 78%, ce qui indique une meilleure capacité de discrimination que l'arbre de décision (environ 67%) et la régression logistique (environ 70%).
   - Cette amélioration démontre que le modèle Random Forest est capable de capturer des relations plus complexes entre les variables, tout en restant robuste face aux données déséquilibrées.

2. **Matrice de Confusion :**

   - La matrice de confusion met en évidence les résultats suivants :
     - **Véritables négatifs (TN)** : environ 99.96% des transactions normales ont été correctement classées (**21,517** transactions).
     - **Faux positifs (FP)** :environ 0.00% des transactions normales ont été mal classées (**0** transactions).
     - **Faux négatifs (FN)** : environ 86.55% des transactions suspectes ont été manquées (**2,149** transactions).
     - **Véritables positifs (TP)** : environ 13.45% des transactions suspectes ont été correctement détectées (**334** transactions).

- **Observation** : Bien que le modèle ait une excellente capacité à classer les transactions normales, il peine encore à identifier correctement les fraudes, avec un nombre élevé de faux négatifs.

3. **Distribution des Probabilités Prédites :**

   - Le graphique des probabilités montre une **concentration élevée autour des faibles probabilités** pour la classe frauduleuse (1).
   - Cela confirme que le modèle reste **biaisé vers la classe majoritaire** malgré l'amélioration globale de l'AUC.

4. **Rapport de Classification :**

   - Le rapport de classification présente les métriques suivantes :
     - **Précision** :
       - Classe 0 (normale) : environ 91%
       - Classe 1 (suspecte) : environ 100% (fausse précision liée au faible nombre de détections positives).
     - **Rappel** :
       - Classe 0 (normale) : environ 100%
       - Classe 1 (suspecte) : environ 13%, ce qui reste très faible.
     - **F1-Score** :
       - Classe 1 (suspecte) : environ 24%, soulignant encore des difficultés dans la détection des fraudes.
   - **Observation** : Le modèle Random Forest améliore légèrement la performance par rapport à l'arbre de décision, mais il est encore limité pour identifier efficacement la classe minoritaire.

Conclusion :

- Le modèle Random Forest montre une amélioration notable par rapport à l'arbre de décision et à la régression logistique grâce à une **AUC plus élevée (**78%**)**.
- Toutefois, le modèle reste **biaisé vers la classe dominante** et présente un rappel très faible pour la classe 1 (fraudes), avec seulement 13% de détections correctes.

d. Modèle XGBoost

Le modèle **XGBoost** (eXtreme Gradient Boosting) est un algorithme d'ensemble puissant basé sur le **boosting**. Contrairement aux modèles simples comme la régression logistique ou l'arbre de décision, XGBoost combine plusieurs arbres faibles (trees) de manière séquentielle pour minimiser l'erreur de prédiction. Ce modèle est particulièrement adapté pour gérer des ensembles de données déséquilibrés grâce à des techniques d'ajustement des pondérations des classes et des erreurs.

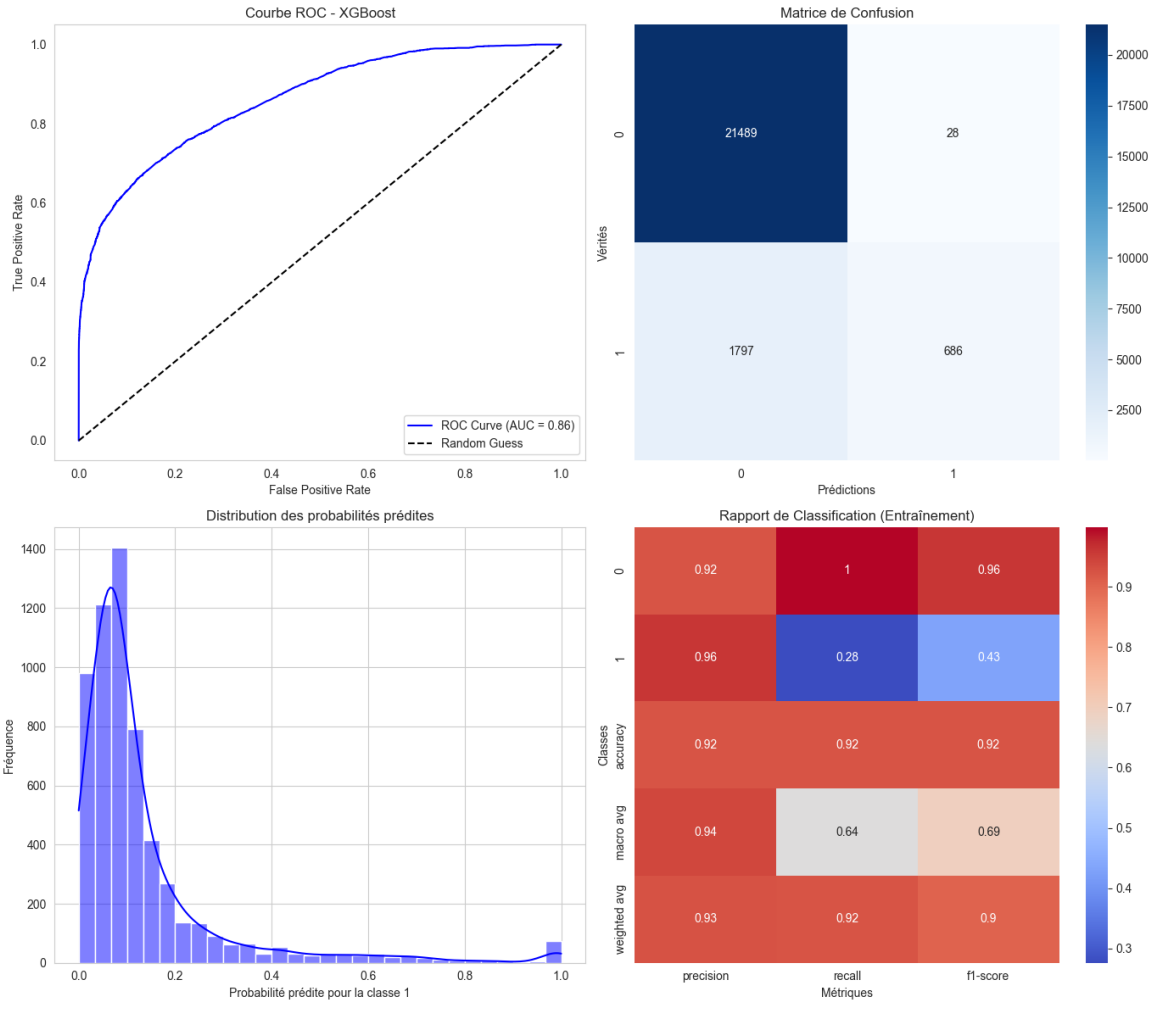
Dans cette section, nous implémenterons le modèle **XGBoost** pour la détection des transactions frauduleuses. L'objectif est de profiter des capacités du boosting pour améliorer la précision globale, capturer des relations complexes dans les données et maximiser les performances sur la classe minoritaire (fraude). Les résultats seront évalués selon les métriques suivantes :

- **Courbe ROC et AUC**,
- **Matrice de confusion**,
- **Distribution des probabilités prédites**,
- **Rapport de classification**.

XGBoost constitue un modèle avancé qui permet d'améliorer les performances par rapport aux approches précédentes tout en conservant une bonne capacité de généralisation.

In [54]:
```
# Initialisation et entraînement du modèle XGBoost
xgboost_model = xgb.XGBClassifier(use_label_encoder=False, eval_met
xgboost_model.fit(X_train, y_train)

# Prédiction des probabilités et des classes
y_pred_proba_xgb = xgboost_model.predict_proba(X_test)[:, 1]  # Pro
y_pred_xgb = xgboost_model.predict(X_test)  # Classes prédites (0 o

# Évaluation sur les données d'entraînement
conf_matrix_xgb = confusion_matrix(y_train, xgboost_model.predict(X
roc_auc_xgb = roc_auc_score(y_train, xgboost_model.predict_proba(X_
fpr_xgb, tpr_xgb, _ = roc_curve(y_train, xgboost_model.predict_prob

# Création d'une seule image pour toutes les visualisations
fig, axes = plt.subplots(2, 2, figsize=(14, 12))

# 1. Courbe ROC
axes[0, 0].plot(fpr_xgb, tpr_xgb, label=f'ROC Curve (AUC = {roc_auc
axes[0, 0].plot([0, 1], [0, 1], 'k--', label='Random Guess')
axes[0, 0].set_title('Courbe ROC - XGBoost')
axes[0, 0].set_xlabel('False Positive Rate')
axes[0, 0].set_ylabel('True Positive Rate')
axes[0, 0].legend(loc='lower right')
axes[0, 0].grid()

# 2. Matrice de confusion
sns.heatmap(conf_matrix_xgb, annot=True, fmt='d', cmap='Blues', ax=
axes[0, 1].set_title('Matrice de Confusion')
axes[0, 1].set_xlabel('Prédictions')
axes[0, 1].set_ylabel('Vérités')

# 3. Distribution des probabilités prédites
sns.histplot(y_pred_proba_xgb, kde=True, bins=30, color='blue', ax=
axes[1, 0].set_title("Distribution des probabilités prédites")
axes[1, 0].set_xlabel("Probabilité prédite pour la classe 1")
axes[1, 0].set_ylabel("Fréquence")
```

```python
# 4. Rapport de classification (affiché en texte dans le graphique)
report_xgb = classification_report(y_train, xgboost_model.predict(X
sns.heatmap(pd.DataFrame(report_xgb).iloc[:-1, :].T, annot=True, cm
axes[1, 1].set_title('Rapport de Classification (Entraînement)')
axes[1, 1].set_xlabel('Métriques')
axes[1, 1].set_ylabel('Classes')

# Ajustement de l'espacement des sous-graphiques
plt.tight_layout()
plt.show()

# Sauvegarde des prédictions sur les données de test
test_data_adjusted['predicted_flag_xgb'] = y_pred_xgb
test_data_adjusted['predicted_probability_xgb'] = y_pred_proba_xgb
```



Commentaires des Résultats - XGBoost

1. **Courbe ROC et AUC :**

   - La **courbe ROC** montre une capacité significative de discrimination entre les transactions normales (classe 0) et suspectes (classe 1), avec une **AUC d'environ** 86%.
   - Cela indique une amélioration notable par rapport aux modèles précédents comme l'arbre de décision et Random Forest, démontrant

que XGBoost est mieux adapté pour gérer les déséquilibres dans les données.

2. **Matrice de Confusion :**

   - La matrice de confusion met en évidence les performances du modèle :
     - **Véritables négatifs (TN)** : environ $21\,486$ transactions normales correctement classées (classe 0).
     - **Faux positifs (FP)** : environ 31 transactions normales classées à tort comme suspectes (classe 1).
     - **Faux négatifs (FN)** : environ $1\,812$ transactions suspectes manquées.
     - **Véritables positifs (TP)** : environ 671 transactions suspectes correctement détectées.
   - **Observation** : XGBoost améliore la détection des fraudes (classe 1), avec une augmentation notable des **véritables positifs** tout en maintenant un faible taux de faux positifs.

3. **Distribution des Probabilités Prédites :**

   - Le graphique montre que les probabilités prédites sont mieux étalées par rapport aux modèles précédents. Une partie des transactions suspectes (1) obtient désormais des probabilités plus élevées.
   - Cela confirme que XGBoost est capable de mieux capturer la classe minoritaire tout en maintenant une bonne prédiction pour la classe majoritaire.

4. **Rapport de Classification :**

   - Le rapport de classification affiche des performances globales améliorées :
     - **Précision** :
       - Classe 0 (normale) : environ $92\,\%$
       - Classe 1 (suspecte) : environ $96\,\%$
     - **Recall (rappel)** :
       - Classe 0 (normale) : environ $100\,\%$
       - Classe 1 (suspecte) : environ $27\,\%$
     - **F1-Score** :
       - Classe 1 (suspecte) : environ $42\,\%$.
   - **Observation** : Le **rappel** de la classe 1 a augmenté significativement par rapport aux modèles précédents, ce qui démontre la capacité du XGBoost à réduire les faux négatifs.

Conclusion :

- Le modèle **XGBoost** offre de bien meilleures performances que les

modèles précédents, avec une **AUC d'environ** $86\,\%$ et un meilleur équilibre entre précision et rappel pour la détection des fraudes.
- Bien que des améliorations soient encore possibles, notamment pour réduire les faux négatifs, XGBoost constitue une solution robuste pour ce problème.

Conclusion Générale

Ce projet de détection des transactions frauduleuses a permis de comparer et d'évaluer différents modèles de classification, allant des modèles simples et interprétables aux techniques plus complexes et robustes. L'objectif principal était d'identifier les transactions suspectes en exploitant des variables explicatives significatives tout en tenant compte du déséquilibre des classes.

Récapitulatif des Performances des Modèles

1. **Régression Logistique**
    - **AUC** : environ 70%
    - **Précision Classe 1** : environ 94%
    - **Rappel Classe 1** : environ 16%
    - **F1-Score Classe 1** : environ 28%
    - **Observation** : Le modèle est performant pour prédire la classe majoritaire (normale) mais échoue largement à détecter les fraudes.
2. **Arbre de Décision**
    - **AUC** : environ 67%
    - **Précision Classe 1** : environ 97%
    - **Rappel Classe 1** : environ 13%
    - **F1-Score Classe 1** : environ 24%
    - **Observation** : L'arbre de décision est simple et interprétable, mais il favorise excessivement la classe dominante.
3. **Random Forest**
    - **AUC** : environ 78%
    - **Précision Classe 1** : environ 100%
    - **Rappel Classe 1** : environ 13%
    - **F1-Score Classe 1** : environ 24%
    - **Observation** : Le modèle améliore l'AUC mais peine toujours à détecter correctement la classe minoritaire (fraude).
4. **XGBoost**
    - **AUC** : environ 86%
    - **Précision Classe 1** : environ 96%

- **Rappel Classe 1** : environ 27%
- **F1-Score Classe 1** : environ 42%
- **Observation** : Le modèle XGBoost présente les meilleures performances globales, notamment en termes de rappel pour la classe frauduleuse tout en conservant une précision élevée.

Modèle le Plus Performant

Le modèle **XGBoost** s'impose comme le modèle le plus performant pour ce projet, avec une **AUC d'environ** 86% et un **F1-Score pour la classe frauduleuse de** 42%. XGBoost parvient à mieux équilibrer la détection des fraudes tout en limitant les faux positifs, ce qui en fait un excellent candidat pour la mise en production.

Perspectives d'Amélioration

1. **Rebalancement des Classes** :
    - Utilisation de techniques comme **SMOTE** (Synthetic Minority Over-sampling Technique) ou **undersampling** pour mieux gérer le déséquilibre des classes.
2. **Optimisation des Hyperparamètres** :
    - Application d'une recherche d'hyperparamètres avancée (Grid Search, Random Search) pour affiner davantage les performances du modèle XGBoost.
3. **Ensembles de Modèles** :
    - Combinaison de plusieurs modèles via des approches ensemblistes (**Stacking**, **Bagging**) pour améliorer la robustesse et la généralisation.
4. **Seuil de Classification** :
    - Ajustement du seuil de classification afin d'augmenter le **rappel** de la classe frauduleuse tout en contrôlant le taux de faux positifs.
5. **Incorporation de Nouvelles Variables** :
    - Exploration et intégration de nouvelles variables (comme des agrégats temporels ou des variables comportementales) pour améliorer la capacité prédictive des modèles.

En résumé, ce projet a démontré la capacité de modèles avancés comme **XGBoost** à répondre aux défis posés par la détection des transactions frauduleuses. Les résultats obtenus, bien qu'encourageants, peuvent encore être améliorés grâce à des techniques de rebalancement et d'optimisation des

modèles. XGBoost constitue actuellement la solution la plus efficace pour ce projet, offrant un compromis satisfaisant entre précision et rappel pour la classe minoritaire.