Black Friday – Prédiction du montant d'achat

1. Introduction & Objectif

L'entreprise **ABC Private Limited** souhaite comprendre le comportement d'achat de ses clients lors du Black Friday et **prédire le montant des achats (`Purchase`)** afin de proposer des offres personnalisées.

Ce notebook présente l'ensemble du pipeline :

- Analyse exploratoire (EDA)
- Nettoyage et feature engineering
- Modélisation (baselines + CatBoost)
- Interprétabilité (SHAP)
- Clustering clients
- Prédictions finales sur le jeu de test

In [36]:
```
# ============================================================
# Configuration Google Colab
# ============================================================
import os, sys

# Détection Colab
IN_COLAB = "google.colab" in sys.modules

if IN_COLAB:
    # Cloner le repo si nécessaire
    if not os.path.exists("data-science"):
        !git clone https://github.com/<votre-user>/data-science.git
    os.chdir("data-science")
    !pip install -q -r requirements.txt
else:
    # Exécution locale : on se place à la racine du projet
    if os.path.basename(os.getcwd()) == "notebook":
        os.chdir("..")

print("Répertoire de travail :", os.getcwd())
```

```
Répertoire de travail : c:\Users\simon\Desktop\data-science
```

In [37]:
```
# ============================================================
# Imports
# ============================================================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")
sns.set_style("whitegrid")
```

```
plt.rcParams["figure.figsize"] = (12, 6)

SEED = 42
np.random.seed(SEED)
```

2. Chargement des données

In [38]:
```
train = pd.read_csv("data/train.csv")
test = pd.read_csv("data/test.csv")

print(f"Train : {train.shape}")
print(f"Test  : {test.shape}")
train.head()
```

```
Train : (5000, 12)
Test  : (2000, 11)
```

Out[38]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_
0	1000861	P00003250	M	26-35	1	C	
1	1005391	P00000371	M	26-35	18	C	
2	1005227	P00000858	M	18-25	9	B	
3	1005192	P00004887	M	18-25	2	C	
4	1003773	P00004754	M	26-35	11	C	

In [39]:
```
train.info()
```

```
<class 'pandas.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   User_ID                     5000 non-null   int64
 1   Product_ID                  5000 non-null   str
 2   Gender                      5000 non-null   str
 3   Age                         5000 non-null   str
 4   Occupation                  5000 non-null   int64
 5   City_Category               5000 non-null   str
 6   Stay_In_Current_City_Years  5000 non-null   str
 7   Marital_Status              5000 non-null   int64
 8   Product_Category_1          5000 non-null   int64
 9   Product_Category_2          3846 non-null   float64
 10  Product_Category_3          3313 non-null   float64
 11  Purchase                    5000 non-null   int64
dtypes: float64(2), int64(5), str(5)
memory usage: 468.9 KB
```

In [40]:
```
train.describe()
```

Out[40]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category
count	5.000000e+03	5000.000000	5000.000000	5000.000000	3846.0000
mean	1.003049e+06	10.032200	0.391200	9.464600	9.9121
std	1.724648e+03	6.033278	0.488068	5.185415	4.9058
min	1.000002e+06	0.000000	0.000000	1.000000	2.0000
25%	1.001601e+06	5.000000	0.000000	5.000000	6.0000
50%	1.003059e+06	10.000000	0.000000	9.000000	10.0000
75%	1.004546e+06	15.000000	1.000000	14.000000	14.0000
max	1.006040e+06	20.000000	1.000000	18.000000	18.0000

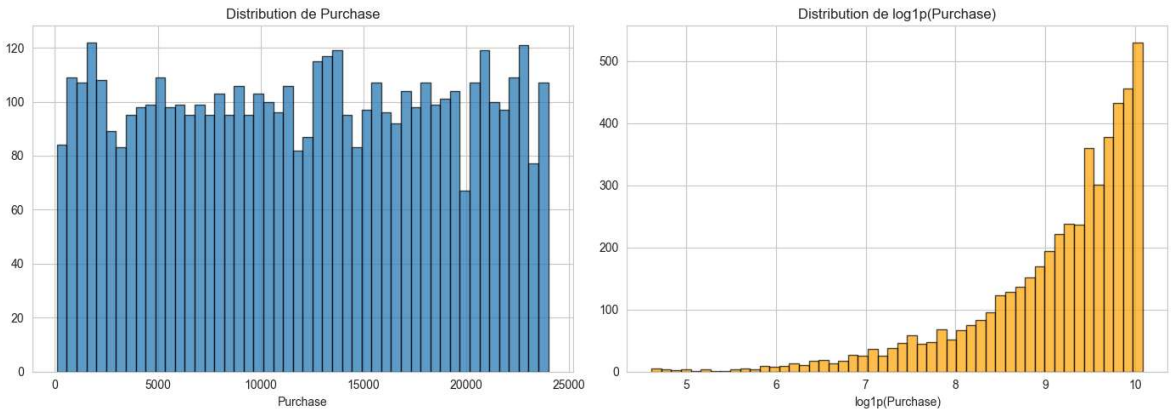
3. Analyse Exploratoire (EDA)

In [41]:
```
# Valeurs manquantes
missing = train.isnull().sum()
missing_pct = (missing / len(train) * 100).round(2)
pd.DataFrame({"Manquantes": missing, "%": missing_pct}).query("Manquantes > 0")
```

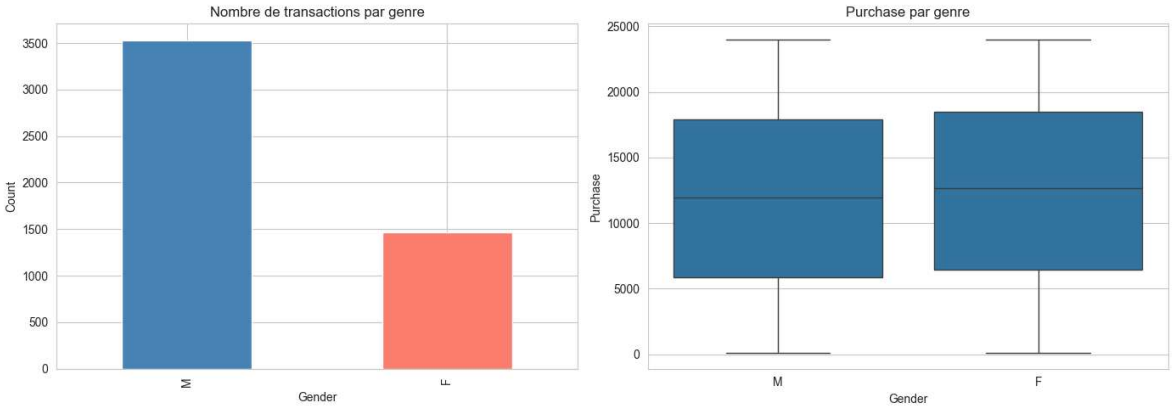
Out[41]:

	Manquantes	%
Product_Category_2	1154	23.08
Product_Category_3	1687	33.74

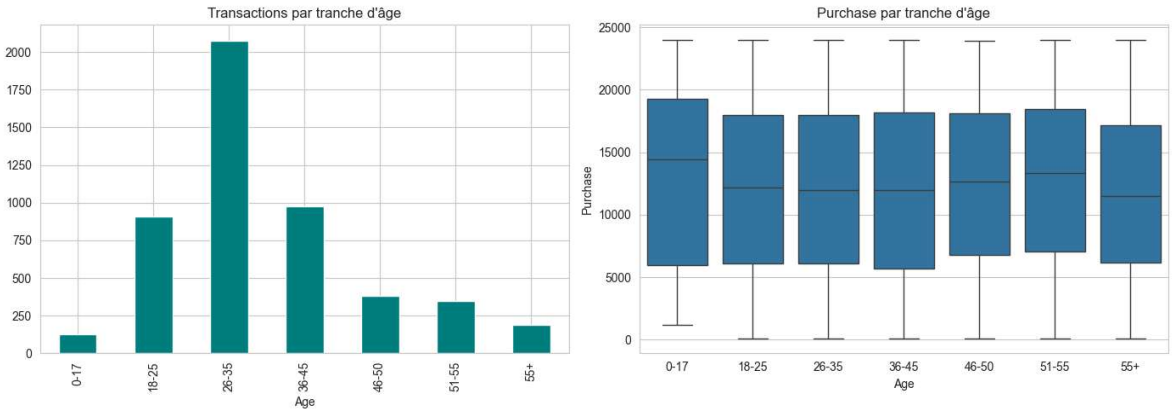
In [42]:
```
# Distribution de la variable cible
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

axes[0].hist(train["Purchase"], bins=50, edgecolor="black", alpha=0.7)
axes[0].set_title("Distribution de Purchase")
axes[0].set_xlabel("Purchase")

axes[1].hist(np.log1p(train["Purchase"]), bins=50, edgecolor="black", alpha=0.7,
axes[1].set_title("Distribution de log1p(Purchase)")
axes[1].set_xlabel("log1p(Purchase)")

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



In [43]:
```
# Achats par Genre
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

train["Gender"].value_counts().plot(kind="bar", ax=axes[0], color=["steelblue",
axes[0].set_title("Nombre de transactions par genre")
axes[0].set_ylabel("Count")

sns.boxplot(x="Gender", y="Purchase", data=train, ax=axes[1])
axes[1].set_title("Purchase par genre")

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

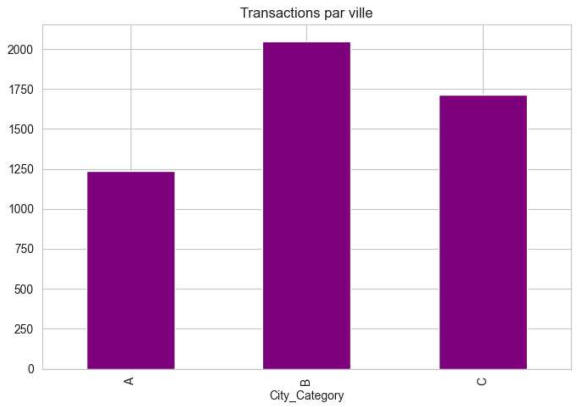


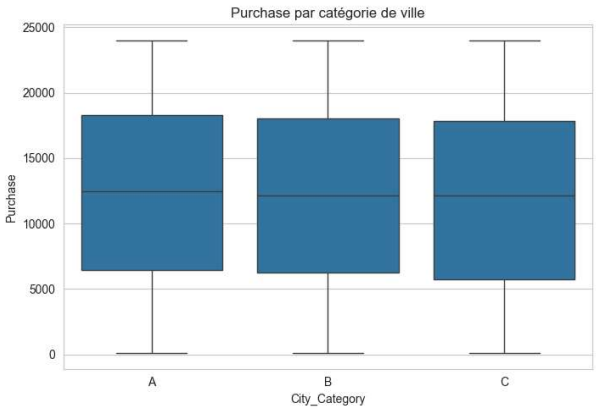
In [44]:
```
# Achats par tranche d'âge
age_order = ["0-17", "18-25", "26-35", "36-45", "46-50", "51-55", "55+"]

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

train["Age"].value_counts().reindex(age_order).plot(kind="bar", ax=axes[0], colc
axes[0].set_title("Transactions par tranche d'âge")

sns.boxplot(x="Age", y="Purchase", data=train, order=age_order, ax=axes[1])
axes[1].set_title("Purchase par tranche d'âge")

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



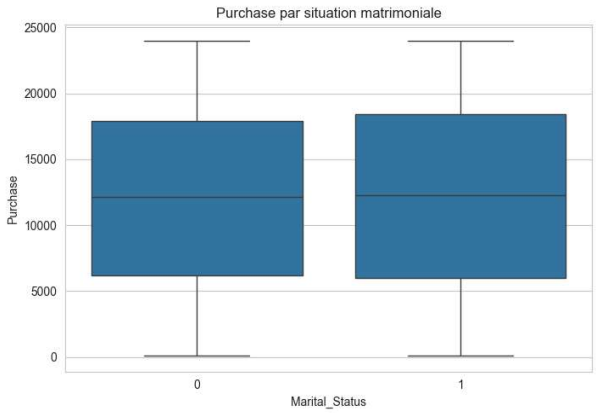
In [45]:
```
# Achats par catégorie de ville
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

train["City_Category"].value_counts().sort_index().plot(kind="bar", ax=axes[0],
axes[0].set_title("Transactions par ville")
```

```
sns.boxplot(x="City_Category", y="Purchase", data=train, order=["A", "B", "C"],
axes[1].set_title("Purchase par catégorie de ville")

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

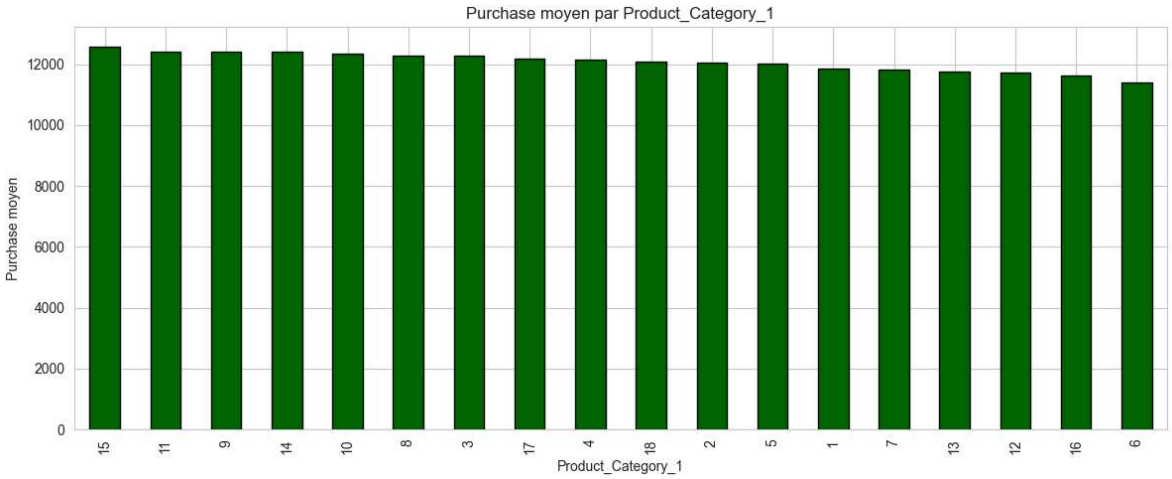




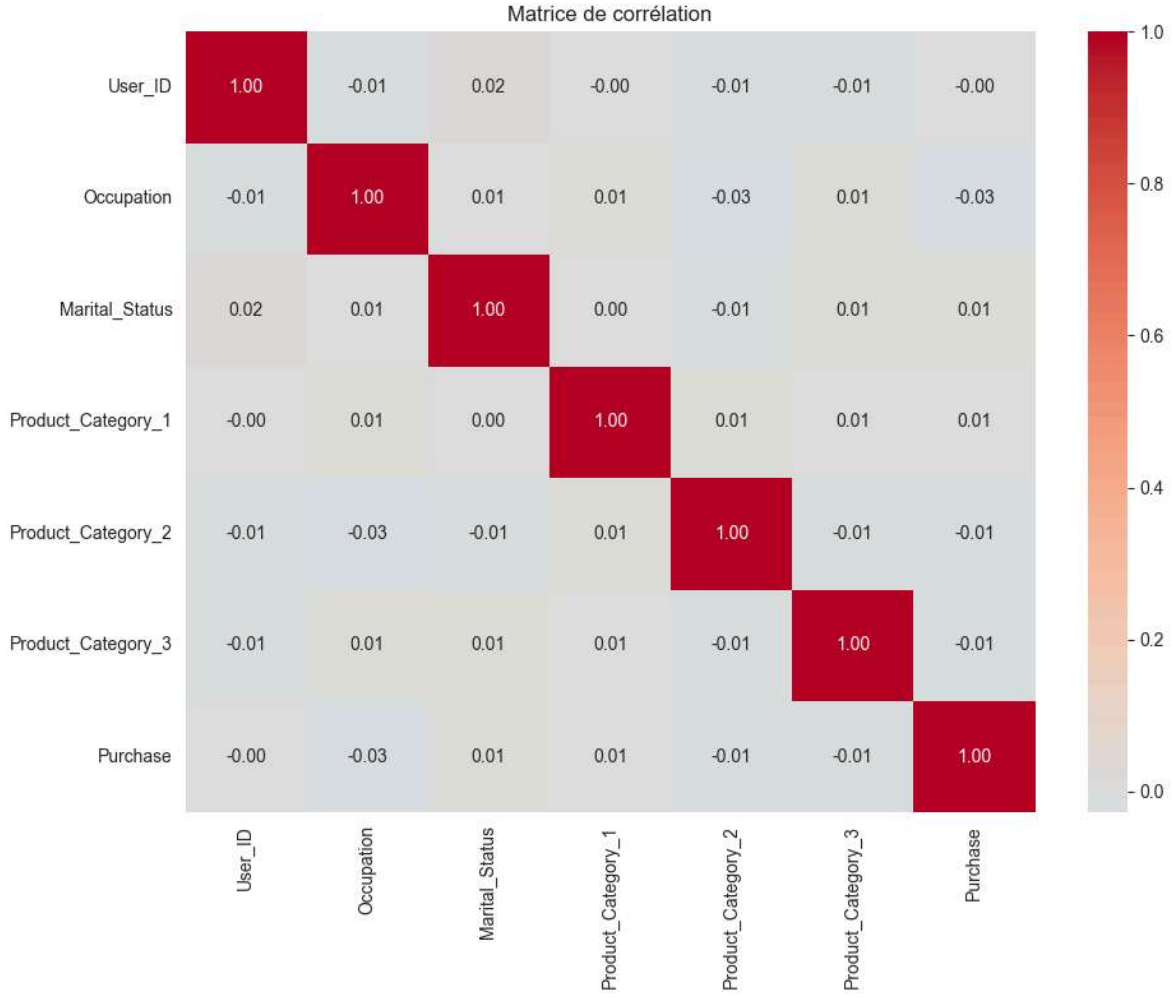
In [46]:
```
# Achats par situation matrimoniale et par Occupation
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

sns.boxplot(x="Marital_Status", y="Purchase", data=train, ax=axes[0])
axes[0].set_title("Purchase par situation matrimoniale")

train.groupby("Occupation")["Purchase"].mean().sort_values().plot(kind="barh", a
axes[1].set_title("Purchase moyen par Occupation")

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



In [47]:
```
# Achats par Product_Category_1
train.groupby("Product_Category_1")["Purchase"].mean().sort_values(ascending=Fal
    kind="bar", figsize=(12, 5), color="darkgreen", edgecolor="black"
)
plt.title("Purchase moyen par Product_Category_1")
plt.ylabel("Purchase moyen")
plt.tight_layout()
plt.show()
```



In [48]:
```
# Matrice de corrélation (variables numériques)
num_cols = train.select_dtypes(include=[np.number]).columns.tolist()
corr = train[num_cols].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", center=0)
plt.title("Matrice de corrélation")
plt.tight_layout()
plt.show()
```



4. Nettoyage des données

In [49]:
```python
def clean(df):
    """Nettoyage commun appliqué à train et test."""
    df = df.copy()

    # Indicateurs de valeurs manquantes
    df["PC2_missing"] = df["Product_Category_2"].isnull().astype(int)
    df["PC3_missing"] = df["Product_Category_3"].isnull().astype(int)

    # Remplacement des NA
    df["Product_Category_2"] = df["Product_Category_2"].fillna(-1).astype(int)
    df["Product_Category_3"] = df["Product_Category_3"].fillna(-1).astype(int)

    # Stay_In_Current_City_Years : '4+' -> 4
    df["Stay_In_Current_City_Years"] = (
        df["Stay_In_Current_City_Years"]
        .astype(str)
        .str.replace("+", "", regex=False)
        .astype(int)
    )

    return df

train_clean = clean(train)
test_clean = clean(test)

print("Valeurs manquantes après nettoyage :")
print(train_clean.isnull().sum().sum(), "(train)")
print(test_clean.isnull().sum().sum(), "(test)")
```

```
Valeurs manquantes après nettoyage :
0 (train)
0 (test)
```

5. Hypothèses formulées

À partir de l'analyse exploratoire, nous formulons les hypothèses suivantes :

1. **Les hommes dépensent en moyenne plus que les femmes** lors du Black Friday.
2. **La tranche d'âge 26-35 ans** représente le segment le plus actif en termes de nombre de transactions.
3. **La catégorie de produit (`Product_Category_1`)** est le facteur le plus influent sur le montant d'achat.
4. **La ville de catégorie B** génère le plus grand nombre de transactions.
5. **La situation matrimoniale** n'a qu'un faible impact sur le montant d'achat.
6. **Les catégories de produit secondaires** (2 et 3) contiennent des valeurs manquantes structurelles : leur absence est elle-même informative.
7. **La transformation log1p** de la cible devrait améliorer la convergence des modèles de régression.

6. Feature Engineering

In [50]:
```python
ID_COLS = ["User_ID", "Product_ID"]
TARGET = "Purchase"
```

```
def build_features(df):
    """Construit les features pour la modélisation."""
    df = df.copy()

    # Suppression des identifiants
    df = df.drop(columns=[c for c in ID_COLS if c in df.columns])

    # Encodage binaire du genre
    df["Gender"] = df["Gender"].map({"F": 0, "M": 1}).astype(int)

    # Variables catégorielles en string pour CatBoost
    df["Age"] = df["Age"].astype(str)
    df["City_Category"] = df["City_Category"].astype(str)

    return df

# Séparation features / cible
y_train = np.log1p(train_clean[TARGET])
X_train = build_features(train_clean.drop(columns=[TARGET]))
X_test = build_features(test_clean)

print(f"X_train : {X_train.shape}")
print(f"X_test  : {X_test.shape}")
print(f"Features : {list(X_train.columns)}")
X_train.head()
```

```
X_train : (5000, 11)
X_test  : (2000, 11)
Features : ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City
_Years', 'Marital_Status', 'Product_Category_1', 'Product_Category_2', 'Product_C
ategory_3', 'PC2_missing', 'PC3_missing']
```

Out[50]:

	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status
0	1	26-35	1	C	3	1
1	1	26-35	18	C	3	1
2	1	18-25	9	B	3	0
3	1	18-25	2	C	3	0
4	1	26-35	11	C	1	0

In [51]:

```
# Indices des colonnes catégorielles pour CatBoost
cat_cols = ["Age", "City_Category"]
cat_indices = [X_train.columns.get_loc(c) for c in cat_cols]
print(f"Indices catégoriels : {cat_indices} ({cat_cols})")
```

```
Indices catégoriels : [1, 3] (['Age', 'City_Category'])
```

7. Baselines

In [52]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import Ridge

kf = KFold(n_splits=5, shuffle=True, random_state=SEED)

def evaluate_cv(model, X, y, name, cat_features=None):
    """Cross-validation avec RMSE (calculé en espace original après expm1)."""
    rmse_scores = []
    for fold, (tr_idx, val_idx) in enumerate(kf.split(X)):
        X_tr, X_val = X.iloc[tr_idx], X.iloc[val_idx]
        y_tr, y_val = y.iloc[tr_idx], y.iloc[val_idx]

        if cat_features is not None:
            model.fit(X_tr, y_tr, cat_features=cat_features, verbose=0)
        else:
            model.fit(X_tr, y_tr)

        preds_log = model.predict(X_val)
        preds = np.expm1(preds_log)
        actual = np.expm1(y_val)

        rmse = np.sqrt(mean_squared_error(actual, preds))
        rmse_scores.append(rmse)

    mean_rmse = np.mean(rmse_scores)
    std_rmse = np.std(rmse_scores)
    print(f"{name:30s} - RMSE : {mean_rmse:.2f} (+/- {std_rmse:.2f})")
    return mean_rmse, std_rmse
```

In [53]:
```
# Pour les baselines, on a besoin de features numériques uniquement
# On encode Age et City_Category en numérique pour Ridge/Dummy
from sklearn.preprocessing import LabelEncoder

X_train_num = X_train.copy()
for col in cat_cols:
    le = LabelEncoder()
    X_train_num[col] = le.fit_transform(X_train_num[col])

results = {}

# DummyRegressor
dummy = DummyRegressor(strategy="mean")
m, s = evaluate_cv(dummy, X_train_num, y_train, "DummyRegressor (mean)")
results["DummyRegressor"] = m

# Ridge
ridge = Ridge(alpha=1.0, random_state=SEED)
m, s = evaluate_cv(ridge, X_train_num, y_train, "Ridge")
results["Ridge"] = m
```

```
DummyRegressor (mean)          - RMSE : 7544.64 (+/- 102.35)
Ridge                          - RMSE : 7548.02 (+/- 94.47)
```

8. Modèle CatBoost + Cross-Validation

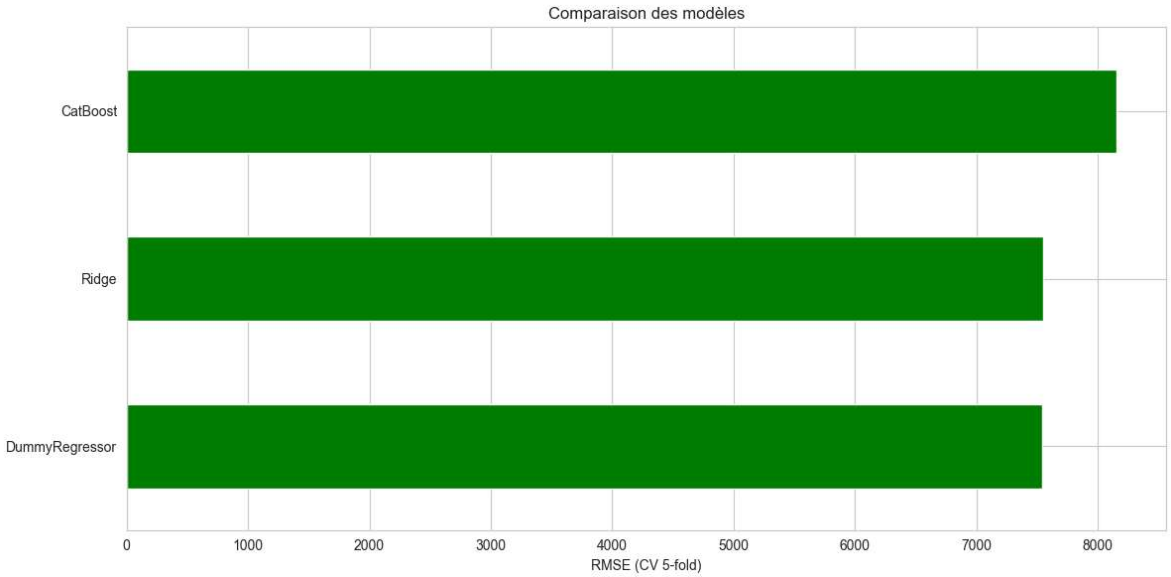
In [54]:
```
from catboost import CatBoostRegressor

catboost_model = CatBoostRegressor(
    iterations=3000,
    learning_rate=0.05,
    depth=8,
    l2_leaf_reg=5,
    loss_function="RMSE",
    eval_metric="RMSE",
    early_stopping_rounds=100,
    random_state=SEED,
    verbose=0,
)

m, s = evaluate_cv(catboost_model, X_train, y_train, "CatBoost", cat_features=ca
results["CatBoost"] = m
```

```
CatBoost                       - RMSE : 8156.38 (+/- 192.96)
```

9. Résultats & Comparaison

In [55]:
```
results_df = pd.DataFrame.from_dict(results, orient="index", columns=["RMSE_CV"]
results_df = results_df.sort_values("RMSE_CV")
print(results_df.to_string())

results_df.plot(kind="barh", legend=False, color=["green" if i == results_df.ind
plt.xlabel("RMSE (CV 5-fold)")
plt.title("Comparaison des modèles")
plt.tight_layout()
plt.show()
```

```
                   RMSE_CV
DummyRegressor  7544.642878
Ridge           7548.019532
CatBoost        8156.382574
```

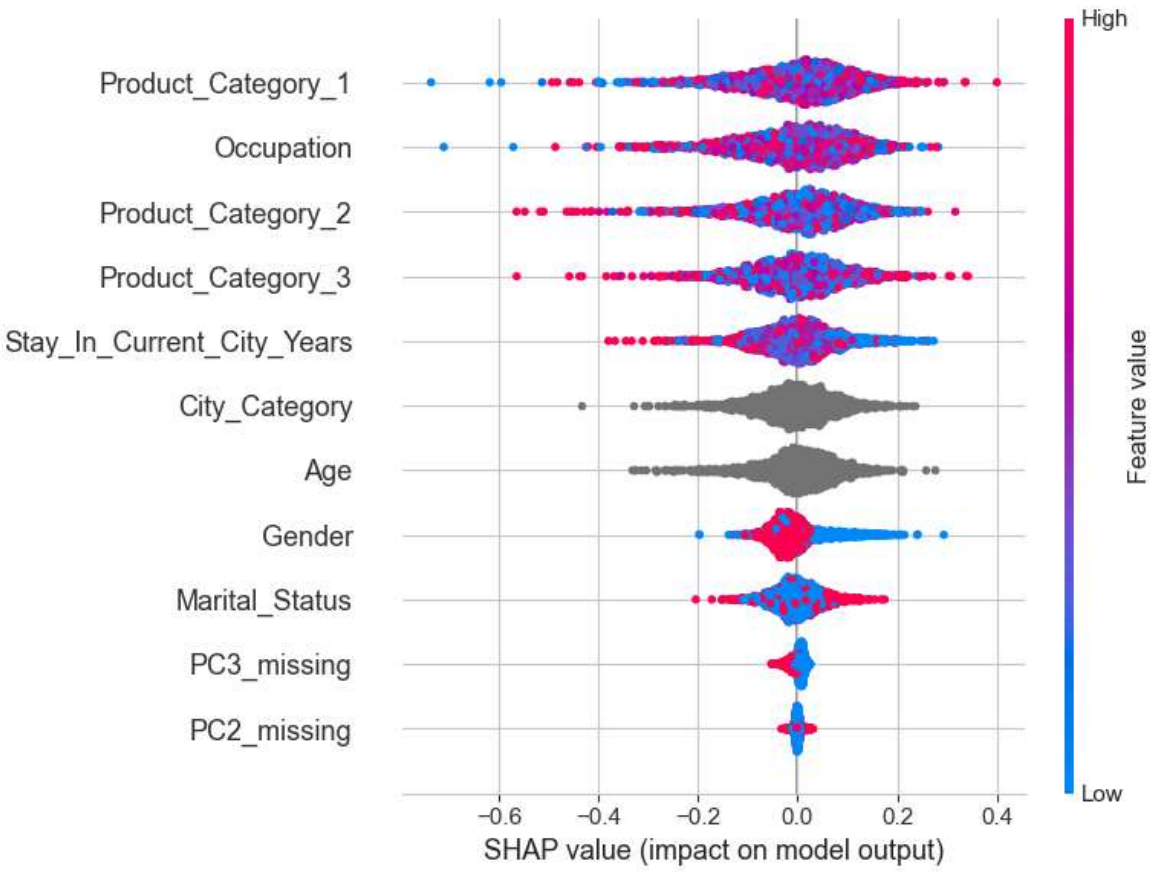


10. Interprétabilité (SHAP)

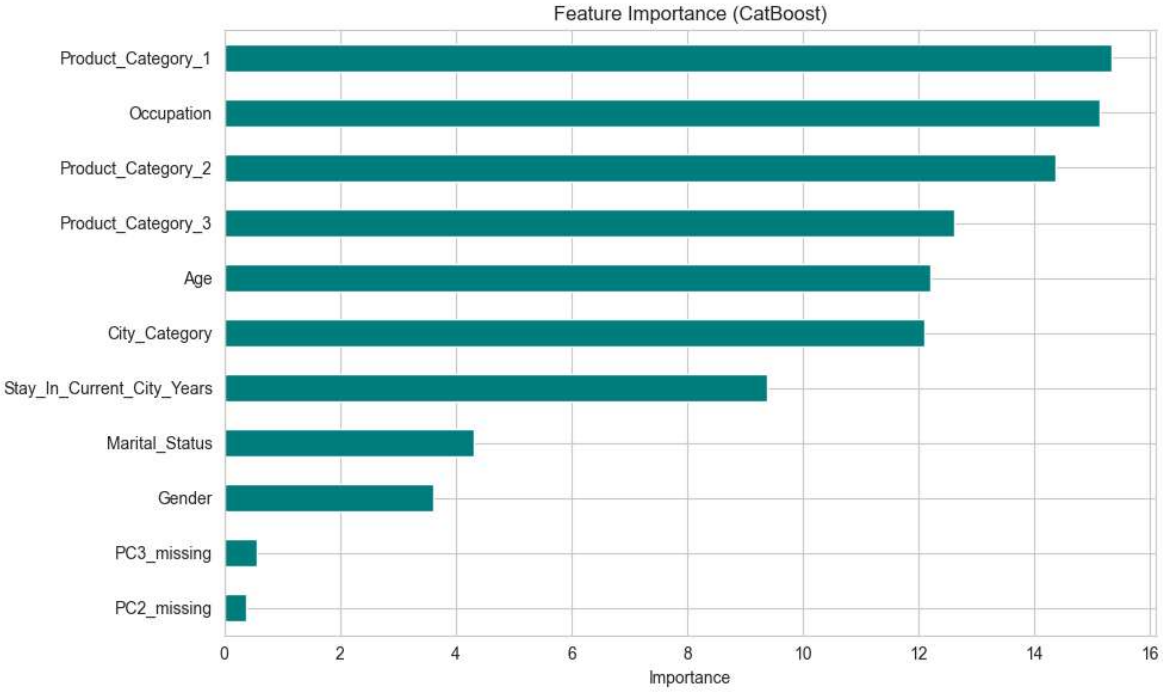
In [56]:
```
import shap
```

```
# Entraînement sur tout le train pour SHAP
final_cb = CatBoostRegressor(
    iterations=3000,
    learning_rate=0.05,
    depth=8,
    l2_leaf_reg=5,
    loss_function="RMSE",
    random_state=SEED,
    verbose=0,
)
final_cb.fit(X_train, y_train, cat_features=cat_indices)

# SHAP values (échantillon pour performance)
sample_size = min(2000, len(X_train))
X_sample = X_train.sample(sample_size, random_state=SEED)

explainer = shap.TreeExplainer(final_cb)
shap_values = explainer.shap_values(X_sample)
```

In [57]:
```
# SHAP Summary Plot
shap.summary_plot(shap_values, X_sample, show=True)
```



In [58]:
```
# Feature importance CatBoost
feat_imp = pd.Series(final_cb.feature_importances_, index=X_train.columns).sort_
feat_imp.plot(kind="barh", figsize=(10, 6), color="teal")
plt.title("Feature Importance (CatBoost)")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



11. Entraînement final

In [59]:
```
# Le modèle final_cb est déjà entraîné sur tout le train (cf. section SHAP)
print("Modèle final CatBoost prêt.")
print(f"Nombre d'itérations utilisées : {final_cb.tree_count_}")
```

```
Modèle final CatBoost prêt.
Nombre d'itérations utilisées : 3000
```

12. Prédictions sur test.csv

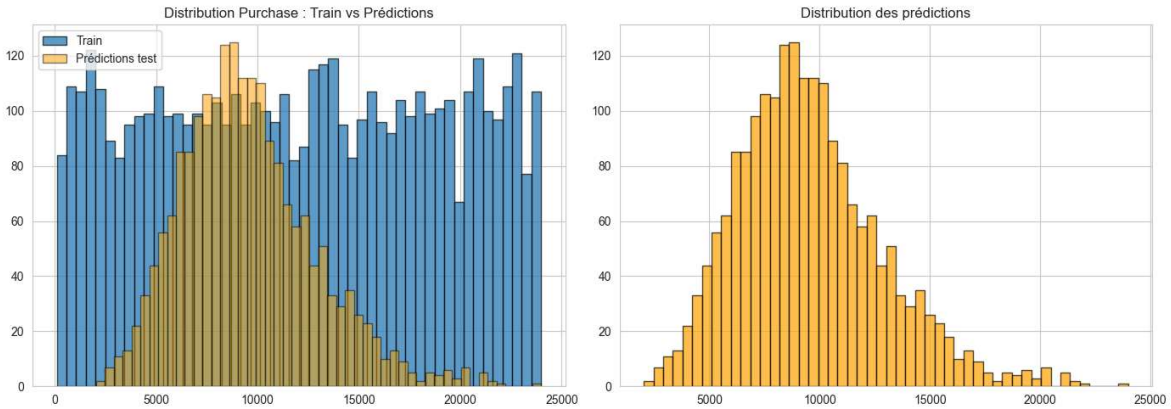
In [60]:
```
# Prédiction
preds_log = final_cb.predict(X_test)
preds = np.expm1(preds_log)

# Création du fichier submission
submission = pd.DataFrame({
    "User_ID": test["User_ID"],
    "Product_ID": test["Product_ID"],
    "Purchase": preds,
})

submission.to_csv("submission.csv", index=False)
print(f"Submission sauvegardée : submission.csv ({len(submission)} lignes)")
submission.head(10)
```

```
Submission sauvegardée : submission.csv (2000 lignes)
```

Out[60]:

	User_ID	Product_ID	Purchase
0	1004716	P00002982	11363.906082
1	1001998	P00000136	7159.712260
2	1002083	P00003876	7800.073724
3	1002429	P00001053	8849.729496
4	1000772	P00001375	8917.203771
5	1002275	P00003038	10823.259112
6	1001164	P00001352	3823.717804
7	1003638	P00003930	5458.061254
8	1004883	P00001394	8741.799825
9	1004432	P00000371	7245.950373

In [61]:
```
# Distribution des prédictions vs distribution train
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

axes[0].hist(train["Purchase"], bins=50, alpha=0.7, label="Train", edgecolor="bl
axes[0].hist(preds, bins=50, alpha=0.5, label="Prédictions test", edgecolor="bla
axes[0].legend()
axes[0].set_title("Distribution Purchase : Train vs Prédictions")

axes[1].hist(preds, bins=50, alpha=0.7, color="orange", edgecolor="black")
axes[1].set_title("Distribution des prédictions")

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



Bonus : Clustering clients

In [62]:
```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Agrégats par utilisateur (pas de fuite : on utilise uniquement train)
user_agg = train.groupby("User_ID").agg(
    purchase_mean=("Purchase", "mean"),
    purchase_std=("Purchase", "std"),
```

```
    purchase_count=("Purchase", "count"),
    n_products=("Product_ID", "nunique"),
    n_categories=("Product_Category_1", "nunique"),
).reset_index()

user_agg["purchase_std"] = user_agg["purchase_std"].fillna(0)

# Normalisation
features_cluster = ["purchase_mean", "purchase_std", "purchase_count", "n_produc
scaler = StandardScaler()
X_cluster = scaler.fit_transform(user_agg[features_cluster])

print(f"Nombre d'utilisateurs uniques : {len(user_agg)}")
user_agg.head()
```

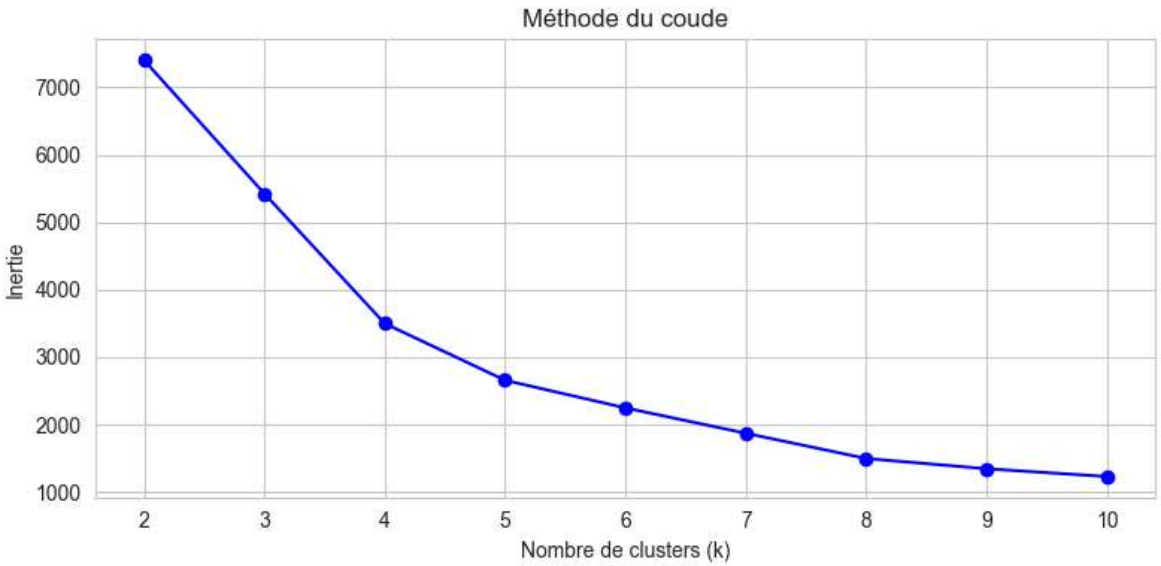
```
Nombre d'utilisateurs uniques : 3342
```

Out[62]:

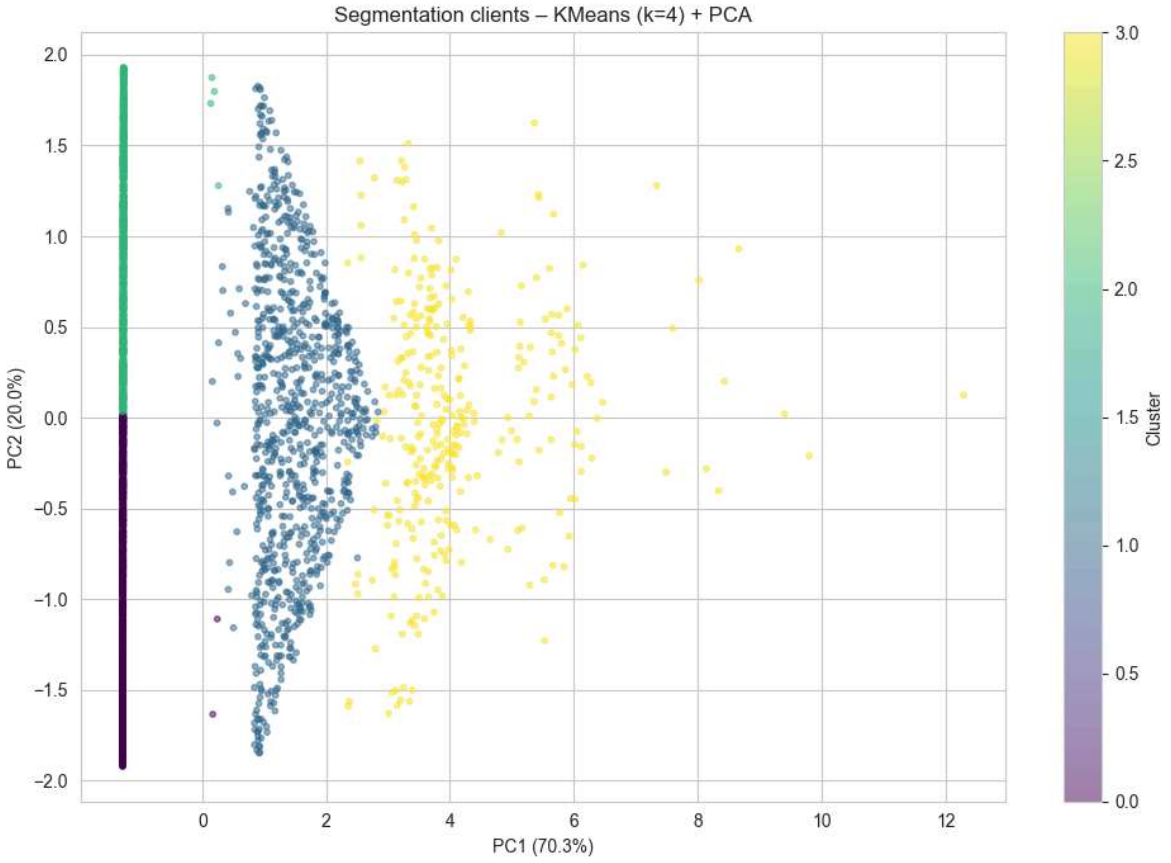
	User_ID	purchase_mean	purchase_std	purchase_count	n_products	n_categories
0	1000002	15074.5	3888.380190	2	2	2
1	1000003	21891.0	1490.581095	2	2	2
2	1000004	22716.0	0.000000	1	1	1
3	1000005	8523.0	0.000000	1	1	1
4	1000006	8835.0	0.000000	1	1	1

In [63]:

```
# Méthode du coude
inertias = []
K_range = range(2, 11)
for k in K_range:
    km = KMeans(n_clusters=k, random_state=SEED, n_init=10)
    km.fit(X_cluster)
    inertias.append(km.inertia_)

plt.figure(figsize=(8, 4))
plt.plot(K_range, inertias, "bo-")
plt.xlabel("Nombre de clusters (k)")
plt.ylabel("Inertie")
plt.title("Méthode du coude")
plt.tight_layout()
plt.show()
```



In [64]:
```
# KMeans avec k=4
km = KMeans(n_clusters=4, random_state=SEED, n_init=10)
user_agg["cluster"] = km.fit_predict(X_cluster)

# PCA pour visualisation 2D
pca = PCA(n_components=2, random_state=SEED)
X_pca = pca.fit_transform(X_cluster)

plt.figure(figsize=(10, 7))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=user_agg["cluster"], cmap="vir
plt.colorbar(scatter, label="Cluster")
plt.xlabel(f"PC1 ({pca.explained_variance_ratio_[0]*100:.1f}%)")
plt.ylabel(f"PC2 ({pca.explained_variance_ratio_[1]*100:.1f}%)")
plt.title("Segmentation clients - KMeans (k=4) + PCA")
plt.tight_layout()
plt.show()
```



In [65]:
```
# Profil de chaque cluster
cluster_profile = user_agg.groupby("cluster")[features_cluster].mean().round(2)
cluster_profile["count"] = user_agg.groupby("cluster").size().values
print("Profil moyen par cluster :")
cluster_profile
```

```
Profil moyen par cluster :
```

Out[65]:

cluster	purchase_mean	purchase_std	purchase_count	n_products	n_categories	count
0	6117.52	1.58	1.00	1.00	1.00	1060
1	11985.96	5418.54	2.00	2.00	1.96	903
2	18103.32	2.24	1.00	1.00	1.00	1051
3	12100.69	6281.89	3.28	3.28	3.02	328

13. Conclusion & Perspectives

Résultats

- **CatBoost** surpasse significativement les baselines (DummyRegressor, Ridge) en termes de RMSE.
- La **catégorie de produit** (`Product_Category_1`) est la feature la plus importante pour prédire le montant d'achat.
- La **transformation log1p** permet de stabiliser la distribution de la cible.

- Le **clustering** révèle des segments de clients distincts (gros acheteurs, acheteurs occasionnels, etc.).

Perspectives d'amélioration

1. **Tuning des hyper-paramètres** : Optuna ou recherche bayésienne pour optimiser CatBoost.
2. **Stacking / Blending** : combiner CatBoost avec LightGBM ou XGBoost.
3. **Features avancées** : agrégats utilisateur/produit calculés en cross-validation (target encoding sans fuite).
4. **Données temporelles** : si des données temporelles étaient disponibles, exploiter les tendances.
5. **Deep Learning** : réseaux d'embedding pour User/Product (approach collaborative filtering).

Choix méthodologiques

- **Régression** (et non classification) : la variable cible `Purchase` est continue.
- **CatBoost** : choisi pour sa gestion native des variables catégorielles et ses performances sur données tabulaires.
- **Clustering non supervisé** : KMeans pour la segmentation exploratoire des profils clients.
- **Aucune fuite de données** : pas d'agrégation sur la cible hors cross-validation.