Étude de Cas Complète: Machine Learning Bancaire

Contexte

Institution: UniBank Haiti
Département: Direction des Risques et Analytics
Mission: Développer des modèles prédictifs pour améliorer la gestion des risques et l'efficacité commerciale

---

Scénario Global

UniBank Haiti souhaite moderniser son approche analytique en intégrant des modèles de Machine Learning pour trois cas d'usage prioritaires:

1. **Scoring de crédit** - Améliorer la prédiction des défauts de paiement
2. **Détection de fraude** - Identifier les transactions suspectes en temps réel
3. **Prédiction de churn** - Anticiper les départs clients pour les retenir

Vous disposez de données historiques sur 50,000 clients et 2 millions de transactions sur les 24 derniers mois.

CAS 1: Scoring de Crédit

Contexte

La banque reçoit environ 500 demandes de prêts par mois. Le processus actuel basé sur des règles simples génère un taux de défaut de 8%. L'objectif est de réduire ce taux à 5% tout en maintenant le volume d'acceptation.

Données Disponibles

```
# Structure du dataset demandes de crédit
colonnes_credit = {
    'id_demande': 'Identifiant unique',
    'age': 'Âge du demandeur',
    'revenu_mensuel': 'Revenu en HTG',
    'anciennete_emploi': 'Mois dans l\'emploi actuel',
    'type_emploi': 'Salarié/Indépendant/Sans emploi',
    'proprietaire': 'Propriétaire du logement (0/1)',
    'nb_dependants': 'Nombre de personnes à charge',
    'dette_existante': 'Encours de dettes actuelles',
    'nb_credits_passes': 'Historique de crédits',
    'nb_retards_12m': 'Retards de paiement sur 12 mois',
    'montant_demande': 'Montant du prêt demandé',
    'duree_mois': 'Durée demandée',
    'objet_pret': 'Consommation/Auto/Immobilier/Entreprise',
    'defaut_12m': 'A fait défaut dans les 12 mois (cible)'
}
```

Questions

Question 1.1: Préparation des Données Décrivez le processus complet de préparation des données pour un modèle de scoring.

Réponse Attendue:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

def preparer_donnees_scoring(df):
    """Pipeline de préparation pour scoring de crédit"""

    # 1. Analyse des valeurs manquantes
    print("Valeurs manquantes:")
    print(df.isnull().sum())

    # 2. Traitement des valeurs manquantes
    # Numériques: médiane (robuste aux outliers)
    for col in ['revenu_mensuel', 'anciennete_emploi']:
        df[col].fillna(df[col].median(), inplace=True)

    # Catégorielles: mode ou catégorie "Inconnu"
    df['type_emploi'].fillna('Inconnu', inplace=True)

    # 3. Feature Engineering
    # Ratio dette/revenu
    df['ratio_dette_revenu'] = df['dette_existante'] / (df['revenu_mensuel'] + 1)

    # Capacité de remboursement mensuelle
    df['mensualite_estimee'] = df['montant_demande'] / df['duree_mois']
    df['ratio_mensualite_revenu'] = df['mensualite_estimee'] / (df['revenu_mensuel'] + 1)

    # Score de stabilité
    df['score_stabilite'] = (df['anciennete_emploi'] / 12) + df['proprietaire'] * 2

    # 4. Encodage des variables catégorielles
    # One-hot pour type_emploi (nominale)
    df = pd.get_dummies(df, columns=['type_emploi', 'objet_pret'], drop_first=True)

    # 5. Suppression des identifiants
    df.drop(columns=['id_demande'], inplace=True)

    # 6. Séparation features/target
    X = df.drop(columns=['defaut_12m'])
    y = df['defaut_12m']

    # 7. Split train/test (stratifié pour garder la proportion de défauts)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y
    )

    # 8. Standardisation (fit sur train uniquement!)
```

```
    scaler = StandardScaler()
    colonnes_num = ['age', 'revenu_mensuel', 'anciennete_emploi',
                    'dette_existante', 'montant_demande', 'ratio_dette_revenu',
                    'ratio_mensualite_revenu', 'score_stabilite']

    X_train[colonnes_num] = scaler.fit_transform(X_train[colonnes_num])
    X_test[colonnes_num] = scaler.transform(X_test[colonnes_num])

    return X_train, X_test, y_train, y_test, scaler
```

Points clés: - Toujours analyser les valeurs manquantes avant traitement - Feature engineering spécifique au domaine (ratios financiers) - Encoder correctement selon le type de variable - Fit du scaler UNIQUEMENT sur le train pour éviter le data leakage - Stratification pour préserver la distribution de la variable cible

---

Question 1.2: Construction du Modèle Construisez un modèle de régression logistique et interprétez les coefficients.

Réponse Attendue:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, classification_report
import numpy as np

def construire_modele_scoring(X_train, X_test, y_train, y_test):
    """Construction et évaluation du modèle de scoring"""

    # 1. Entraînement
    model = LogisticRegression(random_state=42, max_iter=1000)
    model.fit(X_train, y_train)

    # 2. Prédictions
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # 3. Métriques
    auc = roc_auc_score(y_test, y_proba)
    gini = 2 * auc - 1

    print(f"AUC-ROC: {auc:.3f}")
    print(f"Gini: {gini:.3f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # 4. Interprétation des coefficients
    print("\nOdds Ratios (facteurs de risque):")
    coefficients = pd.DataFrame({
        'Variable': X_train.columns,
        'Coefficient': model.coef_[0],
        'Odds_Ratio': np.exp(model.coef_[0])
    }).sort_values('Odds_Ratio', ascending=False)

    print(coefficients.head(10))
```

```
    return model, coefficients

# Interprétation des odds ratios:
# OR > 1: Augmente le risque de défaut
# OR < 1: Diminue le risque de défaut
# OR = 1: Pas d'effet

# Exemple:
# nb_retards_12m: OR = 2.5 → Chaque retard multiplie le risque par 2.5
# proprietaire: OR = 0.6 → Être propriétaire réduit le risque de 40%
```

Points clés: - La régression logistique est privilégiée en banque pour son interprétabilité - Les odds ratios permettent d'expliquer les décisions aux clients et régulateurs - Gini coefficient est la métrique standard en scoring bancaire

---

Question 1.3: Conversion en Score Convertissez les probabilités en un score de 300 à 850.

Réponse Attendue:

```
def probabilite_vers_score(proba, base_score=600, pdo=20):
    """
    Convertit une probabilité de défaut en score de crédit

    Paramètres:
    - proba: Probabilité de défaut (0-1)
    - base_score: Score pour odds = 1 (50% de probabilité)
    - pdo: Points to Double the Odds (combien de points pour doubler le risque)

    Formule: Score = base_score - pdo × log2(odds)
    """

    # Éviter division par zéro
    proba = np.clip(proba, 0.001, 0.999)

    # Calculer les odds
    odds = proba / (1 - proba)

    # Calculer le score
    score = base_score - pdo * np.log2(odds)

    # Borner entre 300 et 850
    score = np.clip(score, 300, 850)

    return score

# Application
df['PD'] = model.predict_proba(X)[:, 1]
df['Score'] = probabilite_vers_score(df['PD'])

# Grille de décision
def decision_credit(score):
```

```
    if score >= 750:
        return 'Accepté - Taux préférentiel'
    elif score >= 650:
        return 'Accepté - Taux standard'
    elif score >= 550:
        return 'Accepté - Taux majoré + garantie'
    else:
        return 'Refusé'

df['Decision'] = df['Score'].apply(decision_credit)

# Analyse par tranche de score
print("Taux de défaut par tranche de score:")
df['Tranche_Score'] = pd.cut(df['Score'],
                             bins=[300, 500, 600, 700, 800, 850],
                             labels=['300-500', '500-600', '600-700', '700-800', '800-850
print(df.groupby('Tranche_Score')['defaut_12m'].mean())
```

CAS 2: Détection de Fraude

Contexte

La banque traite environ 100,000 transactions par jour. Le taux de fraude historique est de 0.5%. Les fraudes non détectées coûtent en moyenne 15,000 HTG par incident.

Données Disponibles

```
colonnes_transactions = {
    'id_transaction': 'Identifiant unique',
    'id_client': 'Identifiant client',
    'date_heure': 'Timestamp de la transaction',
    'montant': 'Montant en HTG',
    'type_transaction': 'Retrait/Transfert/Paiement/Dépôt',
    'canal': 'Agence/ATM/Mobile/Internet',
    'localisation': 'Coordonnées GPS',
    'pays': 'Pays de la transaction',
    'device_id': 'Identifiant de l\'appareil',
    'fraude': 'Transaction frauduleuse (0/1)'
}
```

Questions

Question 2.1: Feature Engineering pour la Fraude Créez des features pertinentes pour détecter les comportements frauduleux.

Réponse Attendue:

```
def creer_features_fraude(df):
    """Feature engineering pour détection de fraude"""

    # Convertir en datetime
    df['date_heure'] = pd.to_datetime(df['date_heure'])
```

```python
    # 1. Features temporelles
    df['heure'] = df['date_heure'].dt.hour
    df['jour_semaine'] = df['date_heure'].dt.dayofweek
    df['est_weekend'] = (df['jour_semaine'] >= 5).astype(int)
    df['est_nuit'] = ((df['heure'] < 6) | (df['heure'] > 22)).astype(int)

    # 2. Features de montant
    # Statistiques par client
    client_stats = df.groupby('id_client')['montant'].agg(['mean', 'std', 'max'])
    client_stats.columns = ['montant_moyen_client', 'montant_std_client', 'montant_max_cli
    df = df.merge(client_stats, on='id_client', how='left')

    # Ratio par rapport à la moyenne du client
    df['ratio_montant_moyenne'] = df['montant'] / (df['montant_moyen_client'] + 1)

    # Z-score du montant
    df['zscore_montant'] = (df['montant'] - df['montant_moyen_client']) / (df['montant_std

    # 3. Features de fréquence
    # Nombre de transactions par jour par client
    df['date'] = df['date_heure'].dt.date
    tx_par_jour = df.groupby(['id_client', 'date']).size().reset_index(name='nb_tx_jour')
    df = df.merge(tx_par_jour, on=['id_client', 'date'], how='left')

    # 4. Features de localisation
    # Distance par rapport à la localisation habituelle
    loc_habituelle = df.groupby('id_client')['localisation'].agg(
        lambda x: x.mode()[0] if len(x.mode()) > 0 else x.iloc[0]
    )
    df['loc_habituelle'] = df['id_client'].map(loc_habituelle)

    # Pays inhabituel
    df['pays_risque'] = df['pays'].isin(['Country_A', 'Country_B']).astype(int)
    df['pays_different'] = (df['pays'] != 'Haiti').astype(int)

    # 5. Features de canal
    canal_risque = {'Internet': 2, 'Mobile': 1, 'ATM': 1, 'Agence': 0}
    df['risque_canal'] = df['canal'].map(canal_risque)

    # 6. Features de device
    # Nombre de devices par client
    devices_par_client = df.groupby('id_client')['device_id'].nunique()
    df['nb_devices_client'] = df['id_client'].map(devices_par_client)

    # Nouveau device
    premier_device = df.groupby('id_client')['device_id'].first()
    df['nouveau_device'] = (df['device_id'] != df['id_client'].map(premier_device)).astype

    return df
```

Points clés: - Les features temporelles sont cruciales (transactions de nuit, weekend) - Comparer au comportement habituel du client - Identifier les anomalies géographiques - Surveiller les changements de device

---

Question 2.2: Gestion du Déséquilibre de Classes Comment gérer le fait que seulement 0.5% des transactions sont frauduleuses?

Réponse Attendue:

```python
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_recall_curve, average_precision_score

def modele_fraude_desequilibre(X_train, X_test, y_train, y_test):
    """Modèle de détection de fraude avec gestion du déséquilibre"""

    print(f"Distribution originale:")
    print(f"Non-fraude: {(y_train == 0).sum()} ({(y_train == 0).mean():.2%})")
    print(f"Fraude: {(y_train == 1).sum()} ({(y_train == 1).mean():.2%})")

    # MÉTHODE 1: SMOTE (Synthetic Minority Over-sampling)
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

    print(f"\nAprès SMOTE:")
    print(f"Non-fraude: {(y_resampled == 0).sum()}")
    print(f"Fraude: {(y_resampled == 1).sum()}")

    # MÉTHODE 2: Class weights
    model_weighted = RandomForestClassifier(
        n_estimators=100,
        class_weight='balanced',  # Pondère inversement à la fréquence
        random_state=42
    )

    # MÉTHODE 3: Sous-échantillonnage de la classe majoritaire
    rus = RandomUnderSampler(random_state=42)
    X_under, y_under = rus.fit_resample(X_train, y_train)

    # Entraînement avec SMOTE
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_resampled, y_resampled)

    # Prédictions
    y_proba = model.predict_proba(X_test)[:, 1]

    # IMPORTANT: Pour la fraude, on optimise le RECALL
    # Car manquer une fraude coûte plus cher qu'un faux positif

    # Trouver le seuil optimal pour recall >= 90%
    precision, recall, thresholds = precision_recall_curve(y_test, y_proba)

    idx = np.argmax(recall >= 0.90)
    optimal_threshold = thresholds[idx] if idx < len(thresholds) else 0.5

    print(f"\nSeuil optimal pour Recall >= 90%: {optimal_threshold:.3f}")

    y_pred = (y_proba >= optimal_threshold).astype(int)
```

```
    # Métriques
    from sklearn.metrics import confusion_matrix
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    print(f"\nRésultats:")
    print(f"Recall (fraudes détectées): {tp/(tp+fn):.2%}")
    print(f"Precision: {tp/(tp+fp):.2%}")
    print(f"Faux positifs: {fp}")
    print(f"Fraudes manquées: {fn}")

    # Coût métier
    cout_faux_positif = 50  # Coût de vérification manuelle
    cout_fraude_manquee = 15000  # Perte moyenne par fraude

    cout_total = fp * cout_faux_positif + fn * cout_fraude_manquee
    print(f"\nCoût total estimé: {cout_total:,.0f} HTG")

    return model, optimal_threshold
```

Points clés: - SMOTE crée des exemples synthétiques de la classe minoritaire - class_weight='balanced' pondère automatiquement les classes - Pour la fraude, privilégier le recall (détecter toutes les fraudes) - Ajuster le seuil de décision selon les coûts métier

---

Question 2.3: Détection d'Anomalies Utilisez un modèle non supervisé pour détecter les transactions anormales.

Réponse Attendue:

```
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import StandardScaler

def detection_anomalies(df, features):
    """Détection d'anomalies non supervisée"""

    X = df[features].copy()

    # Standardisation
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # 1. Isolation Forest
    # Principe: Les anomalies sont plus faciles à isoler
    iso_forest = IsolationForest(
        contamination=0.01,  # 1% attendu d'anomalies
        random_state=42,
        n_estimators=100
    )
    df['anomalie_iso'] = iso_forest.fit_predict(X_scaled)
    # -1 = anomalie, 1 = normal

    # Score d'anomalie (plus négatif = plus anormal)
```

```
    df['score_anomalie'] = iso_forest.decision_function(X_scaled)

    # 2. Local Outlier Factor
    # Principe: Compare la densité locale à celle des voisins
    lof = LocalOutlierFactor(n_neighbors=20, contamination=0.01)
    df['anomalie_lof'] = lof.fit_predict(X_scaled)

    # 3. Consensus des deux méthodes
    df['anomalie_consensus'] = (
        (df['anomalie_iso'] == -1) & (df['anomalie_lof'] == -1)
    ).astype(int)

    # Résumé
    print("Anomalies détectées:")
    print(f"Isolation Forest: {(df['anomalie_iso'] == -1).sum()}")
    print(f"Local Outlier Factor: {(df['anomalie_lof'] == -1).sum()}")
    print(f"Consensus: {df['anomalie_consensus'].sum()}")

    # Si on a les vrais labels
    if 'fraude' in df.columns:
        print("\nPerformance:")
        from sklearn.metrics import precision_score, recall_score

        for method in ['anomalie_iso', 'anomalie_lof', 'anomalie_consensus']:
            y_pred = (df[method] == -1).astype(int) if method != 'anomalie_consensus' else
            recall = recall_score(df['fraude'], y_pred)
            precision = precision_score(df['fraude'], y_pred)
            print(f"{method}: Recall={recall:.2%}, Precision={precision:.2%}")

    return df
```

CAS 3: Prédiction de Churn

Contexte

La banque perd environ 5% de ses clients chaque année. Acquérir un nouveau client coûte 5 fois plus cher que de retenir un client existant. L'objectif est d'identifier les clients à risque de départ 3 mois à l'avance.

Questions

Question 3.1: Features de Churn Identifiez les indicateurs précurseurs du départ client.

Réponse Attendue:

```
def creer_features_churn(df_clients, df_transactions):
    """Features pour prédiction de churn"""

    # 1. Activité récente
    # Dernière transaction
    derniere_tx = df_transactions.groupby('id_client')['date'].max()
    df_clients['jours_depuis_derniere_tx'] = (
        pd.Timestamp.now() - df_clients['id_client'].map(derniere_tx)
```

```
    ).dt.days

    # Nombre de transactions sur 3 mois
    date_3m = pd.Timestamp.now() - pd.Timedelta(days=90)
    tx_3m = df_transactions[df_transactions['date'] >= date_3m]
    nb_tx_3m = tx_3m.groupby('id_client').size()
    df_clients['nb_tx_3m'] = df_clients['id_client'].map(nb_tx_3m).fillna(0)

    # 2. Tendance d'activité
    # Comparer activité récente vs historique
    date_6m = pd.Timestamp.now() - pd.Timedelta(days=180)
    tx_3m_6m = df_transactions[(df_transactions['date'] >= date_6m) &
                               (df_transactions['date'] < date_3m)]
    nb_tx_3m_6m = tx_3m_6m.groupby('id_client').size()

    df_clients['nb_tx_3m_6m'] = df_clients['id_client'].map(nb_tx_3m_6m).fillna(0)
    df_clients['variation_activite'] = (
        (df_clients['nb_tx_3m'] - df_clients['nb_tx_3m_6m']) /
        (df_clients['nb_tx_3m_6m'] + 1)
    )

    # 3. Variation de solde
    # Solde actuel vs solde il y a 3 mois
    df_clients['variation_solde_3m'] = (
        (df_clients['solde_actuel'] - df_clients['solde_3m_avant']) /
        (df_clients['solde_3m_avant'] + 1)
    )

    # 4. Engagement produits
    df_clients['nb_produits'] = df_clients['nb_produits']
    df_clients['a_credit_actif'] = (df_clients['encours_credit'] > 0).astype(int)
    df_clients['a_epargne'] = (df_clients['solde_epargne'] > 0).astype(int)

    # 5. Interactions service client
    df_clients['nb_reclamations_6m'] = df_clients['nb_reclamations_6m']
    df_clients['nb_appels_6m'] = df_clients['nb_appels_service_6m']

    # 6. Satisfaction (si disponible)
    # df_clients['nps'] = df_clients['nps']

    # 7. Variables démographiques
    df_clients['anciennete_mois'] = df_clients['anciennete_mois']

    return df_clients

# Features indicatives de risque de churn:
# - Baisse d'activité: variation_activite < -0.5
# - Solde en baisse: variation_solde_3m < -0.3
# - Inactivité: jours_depuis_derniere_tx > 30
# - Réclamations: nb_reclamations_6m > 2
# - Peu de produits: nb_produits == 1
```

Question 3.2: Modèle de Churn avec Interprétation Construisez un modèle et expliquez les facteurs de risque.

Réponse Attendue:

```python
from sklearn.ensemble import GradientBoostingClassifier
import shap

def modele_churn_interpretable(X_train, X_test, y_train, y_test):
    """Modèle de churn avec interprétation SHAP"""

    # 1. Modèle Gradient Boosting
    model = GradientBoostingClassifier(
        n_estimators=100,
        max_depth=4,
        learning_rate=0.1,
        random_state=42
    )
    model.fit(X_train, y_train)

    # 2. Évaluation
    y_proba = model.predict_proba(X_test)[:, 1]
    auc = roc_auc_score(y_test, y_proba)
    print(f"AUC-ROC: {auc:.3f}")

    # 3. Importance des features
    importance = pd.DataFrame({
        'feature': X_train.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)

    print("\nTop 10 facteurs de risque:")
    print(importance.head(10))

    # 4. Interprétation SHAP
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_test)

    # Graphique d'importance globale
    shap.summary_plot(shap_values, X_test, plot_type="bar")

    # Graphique de dépendance
    shap.summary_plot(shap_values, X_test)

    return model, importance

def expliquer_prediction_individuelle(model, X_test, idx):
    """Expliquer pourquoi un client spécifique est à risque"""

    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_test.iloc[[idx]])

    print(f"Client {idx}:")
    print(f"Probabilité de churn: {model.predict_proba(X_test.iloc[[idx]])[0, 1]:.1%}")
```

```
    # Facteurs qui augmentent le risque
    contributions = pd.DataFrame({
        'feature': X_test.columns,
        'valeur': X_test.iloc[idx].values,
        'contribution': shap_values[0]
    }).sort_values('contribution', ascending=False)

    print("\nFacteurs de risque (ce client):")
    print(contributions[contributions['contribution'] > 0].head(5))

    print("\nFacteurs protecteurs:")
    print(contributions[contributions['contribution'] < 0].head(5))
```

Question 3.3: Plan d'Action Rétention Proposez un plan d'action basé sur les résultats du modèle.

Réponse Attendue:

```
def plan_retention(df, model, X, seuil_risque=0.3):
    """Plan d'action de rétention basé sur le modèle"""

    # Calculer les probabilités de churn
    df['proba_churn'] = model.predict_proba(X)[:, 1]

    # Segmenter par niveau de risque
    df['niveau_risque'] = pd.cut(
        df['proba_churn'],
        bins=[0, 0.2, 0.4, 0.6, 1.0],
        labels=['Faible', 'Modéré', 'Élevé', 'Critique']
    )

    # Calculer la valeur client (simplifiée)
    df['valeur_client'] = df['revenu_annuel_banque']  # ou LTV

    # Matrice Risque x Valeur
    print("Distribution des clients à risque:")
    print(pd.crosstab(df['niveau_risque'],
                      pd.qcut(df['valeur_client'], 4, labels=['Low', 'Medium', 'High', 'VI

    # Plan d'action par segment
    actions = {
        ('Critique', 'VIP'): {
            'action': 'Contact manager dédié immédiat',
            'offre': 'Conditions exceptionnelles, avantages exclusifs',
            'priorite': 1,
            'budget': 50000
        },
        ('Critique', 'High'): {
            'action': 'Appel centre de relation prioritaire',
            'offre': 'Frais réduits, upgrade de compte',
            'priorite': 2,
            'budget': 20000
        },
```

```
        ('Élevé', 'VIP'): {
            'action': 'Email personnalisé + suivi téléphonique',
            'offre': 'Révision des conditions, nouveaux produits',
            'priorite': 3,
            'budget': 15000
        },
        ('Élevé', 'High'): {
            'action': 'Campagne ciblée multicanal',
            'offre': 'Offre de fidélité, cashback',
            'priorite': 4,
            'budget': 10000
        },
        ('Modéré', '*'): {
            'action': 'Email automatisé de fidélisation',
            'offre': 'Offres standard',
            'priorite': 5,
            'budget': 1000
        },
        ('Faible', '*'): {
            'action': 'Aucune action immédiate',
            'offre': 'Communication régulière',
            'priorite': 6,
            'budget': 0
        }
    }

    # Estimation du ROI
    clients_risque = df[df['proba_churn'] >= seuil_risque]

    cout_acquisition_nouveau = 25000  # Coût acquisition nouveau client
    taux_retention_attendu = 0.30  # 30% des clients à risque retenus

    clients_sauves = len(clients_risque) * taux_retention_attendu
    valeur_sauvee = clients_sauves * clients_risque['valeur_client'].mean()
    cout_retention = len(clients_risque) * 5000  # Coût moyen action

    roi = (valeur_sauvee - cout_retention) / cout_retention * 100

    print(f"\nEstimation ROI:")
    print(f"Clients à risque identifiés: {len(clients_risque)}")
    print(f"Clients potentiellement retenus: {clients_sauves:.0f}")
    print(f"Valeur préservée: {valeur_sauvee:,.0f} HTG")
    print(f"Coût des actions: {cout_retention:,.0f} HTG")
    print(f"ROI estimé: {roi:.0f}%")

    return df
```

Synthèse et Bonnes Pratiques

Checklist Projet ML Bancaire

PRÉPARATION

```
□ Comprendre le problème business
□ Identifier les contraintes réglementaires
□ Définir les métriques de succès
□ Analyser la qualité des données

MODÉLISATION
□ Feature engineering métier
□ Split temporel si données temporelles
□ Gérer le déséquilibre de classes
□ Comparer plusieurs algorithmes
□ Validation croisée

ÉVALUATION
□ Métriques appropriées (AUC, Gini pour scoring)
□ Analyse des erreurs
□ Test sur données out-of-time
□ Stabilité des coefficients

INTERPRÉTATION
□ Importance des features
□ Explication individuelle (SHAP)
□ Documentation des décisions
□ Validation métier

DÉPLOIEMENT
□ Pipeline de scoring reproductible
□ Monitoring des performances
□ Alertes de dérive
□ Plan de recalibration
```

Métriques par Cas d'Usage

Cas d'Usage	Métrique Principale	Métrique Secondaire
Scoring Crédit	Gini / AUC	KS, Taux de défaut par décile
Fraude	Recall	Precision, Coût total
Churn	AUC	Lift, ROI rétention

Rappel: En banque, l'interprétabilité et la conformité réglementaire sont aussi importantes que la performance pure du modèle.