Étude de Cas - Exploratory Data Analysis & Data Wrangling

UniBank Haiti - Analyse du Portefeuille de Crédit

Contexte

Le département des risques de UniBank Haiti vous confie un fichier brut extrait du core banking contenant les données du portefeuille de prêts. Avant toute analyse ou modélisation, vous devez: 1. Explorer et comprendre les données 2. Nettoyer et préparer le dataset 3. Identifier les insights clés 4. Préparer les données pour le scoring

---

Données Disponibles

Fichier: `portefeuille_brut.csv` **(15,000 lignes)**

```
Colonnes:
- id_pret: Identifiant unique du prêt
- id_client: Identifiant client
- nom_client: Nom complet
- date_naissance: Date de naissance (formats mixtes)
- sexe: M/F/Masculin/Feminin/NULL
- date_octroi: Date d'octroi du prêt
- date_echeance: Date d'échéance
- montant_initial: Montant initial du prêt
- solde_restant: Solde restant
- taux_interet: Taux d'intérêt (certains en %, d'autres en décimal)
- secteur_activite: Secteur économique
- agence: Code agence
- revenu_mensuel: Revenu déclaré
- anciennete_emploi: Années dans l'emploi actuel
- nb_credits_anterieurs: Nombre de crédits passés
- score_interne: Score de crédit interne (300-850)
- jours_retard: Nombre de jours de retard
- statut: Actif/Soldé/Défaut/Restructuré
- garantie: Type de garantie
- valeur_garantie: Valeur estimée de la garantie
```

PARTIE 1: Exploration Initiale (EDA)

Exercice 1.1

Écrivez le code Python pour charger les données et obtenir une vue d'ensemble complète.

Solution 1.1

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

```
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# 1. Chargement des données
df = pd.read_csv('portefeuille_brut.csv')

# 2. Vue d'ensemble
print("=" * 50)
print("VUE D'ENSEMBLE DU DATASET")
print("=" * 50)

print(f"\n1. DIMENSIONS: {df.shape[0]} lignes × {df.shape[1]} colonnes")

print("\n2. TYPES DE DONNÉES:")
print(df.dtypes)

print("\n3. APERÇU DES DONNÉES:")
print(df.head(10))

print("\n4. INFORMATIONS DÉTAILLÉES:")
print(df.info())

print("\n5. STATISTIQUES DESCRIPTIVES - Variables Numériques:")
print(df.describe())

print("\n6. STATISTIQUES DESCRIPTIVES - Variables Catégorielles:")
print(df.describe(include='object'))

# 3. Analyse des valeurs manquantes
print("\n7. VALEURS MANQUANTES:")
missing = pd.DataFrame({
    'Colonne': df.columns,
    'Missing': df.isnull().sum(),
    'Pourcentage': (df.isnull().sum() / len(df) * 100).round(2)
})
missing = missing[missing['Missing'] > 0].sort_values('Pourcentage', ascending=False)
print(missing)

# 4. Doublons
print(f"\n8. DOUBLONS:")
print(f"   Lignes dupliquées complètes: {df.duplicated().sum()}")
print(f"   ID prêts dupliqués: {df['id_pret'].duplicated().sum()}")

# 5. Valeurs uniques
print("\n9. CARDINALITÉ (valeurs uniques):")
for col in df.columns:
    print(f"   {col}: {df[col].nunique()} valeurs uniques")
```

Exercice 1.2

Analysez la distribution des variables clés et identifiez les anomalies.

Solution 1.2

```
# 1. Distribution des montants
print("ANALYSE DES MONTANTS")
print("-" * 40)
print(f"Min: {df['montant_initial'].min():,.0f} HTG")
print(f"Max: {df['montant_initial'].max():,.0f} HTG")
print(f"Moyenne: {df['montant_initial'].mean():,.0f} HTG")
print(f"Médiane: {df['montant_initial'].median():,.0f} HTG")
print(f"Écart-type: {df['montant_initial'].std():,.0f} HTG")
print(f"Skewness: {df['montant_initial'].skew():.2f}")
print(f"Kurtosis: {df['montant_initial'].kurtosis():.2f}")

# Visualisation
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

# Histogramme montant
axes[0,0].hist(df['montant_initial'], bins=50, edgecolor='black')
axes[0,0].set_title('Distribution Montant Initial')
axes[0,0].axvline(df['montant_initial'].mean(), color='red', linestyle='--', label='Moyenne')
axes[0,0].axvline(df['montant_initial'].median(), color='green', linestyle='--', label='Médiane')
axes[0,0].legend()

# Box plot montant
axes[0,1].boxplot(df['montant_initial'].dropna())
axes[0,1].set_title('Box Plot Montant')

# Distribution taux
axes[0,2].hist(df['taux_interet'].dropna(), bins=30, edgecolor='black')
axes[0,2].set_title('Distribution Taux d\'Intérêt')

# Score de crédit
axes[1,0].hist(df['score_interne'].dropna(), bins=30, edgecolor='black')
axes[1,0].set_title('Distribution Score Crédit')

# Jours de retard
axes[1,1].hist(df['jours_retard'].dropna(), bins=50, edgecolor='black')
axes[1,1].set_title('Distribution Jours de Retard')

# Répartition par statut
df['statut'].value_counts().plot(kind='bar', ax=axes[1,2])
axes[1,2].set_title('Répartition par Statut')

plt.tight_layout()
plt.savefig('eda_distributions.png')
plt.show()

# 2. Détection des anomalies
print("\nANOMALIES DÉTECTÉES:")
print("-" * 40)
```

```python
# Montants négatifs
negatifs = df[df['montant_initial'] < 0]
print(f"Montants négatifs: {len(negatifs)} ({len(negatifs)/len(df)*100:.2f}%)")

# Taux d'intérêt suspects
taux_suspects = df[(df['taux_interet'] > 100) | (df['taux_interet'] < 0)]
print(f"Taux d'intérêt hors limites: {len(taux_suspects)}")

# Scores hors limites
scores_invalides = df[(df['score_interne'] < 300) | (df['score_interne'] > 850)]
print(f"Scores hors limites [300-850]: {len(scores_invalides)}")

# Dates incohérentes
df['date_octroi_temp'] = pd.to_datetime(df['date_octroi'], errors='coerce')
df['date_echeance_temp'] = pd.to_datetime(df['date_echeance'], errors='coerce')
dates_incoherentes = df[df['date_echeance_temp'] < df['date_octroi_temp']]
print(f"Dates échéance < octroi: {len(dates_incoherentes)}")

# Solde > Montant initial
solde_excessif = df[df['solde_restant'] > df['montant_initial']]
print(f"Solde restant > Montant initial: {len(solde_excessif)}")
```

PARTIE 2: Data Wrangling (Nettoyage)

Exercice 2.1

Nettoyez les valeurs manquantes avec des stratégies appropriées.

Solution 2.1

```python
# Copie pour nettoyage
df_clean = df.copy()

print("TRAITEMENT DES VALEURS MANQUANTES")
print("=" * 50)

# 1. Analyse des patterns de valeurs manquantes
import missingno as msno
msno.matrix(df_clean)
plt.savefig('missing_matrix.png')

# 2. Stratégie par colonne
strategies = {
    'revenu_mensuel': 'median_by_group',  # Médiane par secteur
    'anciennete_emploi': 'median',
    'score_interne': 'median_by_group',  # Médiane par statut
    'valeur_garantie': 'zero_if_no_garantie',
    'garantie': 'mode',
    'sexe': 'mode',
    'secteur_activite': 'mode'
}
```

```python
# 3. Imputation

# Revenu: médiane par secteur
median_revenu_secteur = df_clean.groupby('secteur_activite')['revenu_mensuel'].transform('median')
df_clean['revenu_mensuel'] = df_clean['revenu_mensuel'].fillna(median_revenu_secteur)
# Si encore manquant, médiane globale
df_clean['revenu_mensuel'] = df_clean['revenu_mensuel'].fillna(df_clean['revenu_mensuel'].median())

# Ancienneté: médiane
df_clean['anciennete_emploi'] = df_clean['anciennete_emploi'].fillna(df_clean['anciennete_emploi'].media

# Score: médiane par statut de risque
median_score_statut = df_clean.groupby('statut')['score_interne'].transform('median')
df_clean['score_interne'] = df_clean['score_interne'].fillna(median_score_statut)
df_clean['score_interne'] = df_clean['score_interne'].fillna(df_clean['score_interne'].median())

# Garantie: mode
df_clean['garantie'] = df_clean['garantie'].fillna('Non spécifiée')

# Valeur garantie: 0 si pas de garantie
df_clean['valeur_garantie'] = df_clean.apply(
    lambda row: 0 if row['garantie'] == 'Non spécifiée' else row['valeur_garantie'],
    axis=1
)
df_clean['valeur_garantie'] = df_clean['valeur_garantie'].fillna(0)

# Sexe: mode
df_clean['sexe'] = df_clean['sexe'].fillna(df_clean['sexe'].mode()[0])

# Secteur: mode
df_clean['secteur_activite'] = df_clean['secteur_activite'].fillna(df_clean['secteur_activite'].mode()[

# 4. Créer indicateurs de données manquantes (pour variables importantes)
df_clean['revenu_was_missing'] = df['revenu_mensuel'].isnull().astype(int)
df_clean['score_was_missing'] = df['score_interne'].isnull().astype(int)

# 5. Vérification
print("\nValeurs manquantes après nettoyage:")
print(df_clean.isnull().sum()[df_clean.isnull().sum() > 0])
```

Exercice 2.2

Corrigez les incohérences et standardisez les formats.

Solution 2.2

```python
print("STANDARDISATION ET CORRECTION")
print("=" * 50)

# 1. Standardisation du sexe
print("\nAvant standardisation - Sexe:")
```

```
print(df_clean['sexe'].value_counts())

df_clean['sexe'] = df_clean['sexe'].str.upper().str.strip()
df_clean['sexe'] = df_clean['sexe'].replace({
    'MASCULIN': 'M',
    'FEMININ': 'F',
    'MALE': 'M',
    'FEMALE': 'F'
})

print("\nAprès standardisation - Sexe:")
print(df_clean['sexe'].value_counts())

# 2. Correction des taux d'intérêt (certains en %, d'autres en décimal)
print("\n\nCorrection des taux d'intérêt:")
print(f"Avant - Min: {df_clean['taux_interet'].min()}, Max: {df_clean['taux_interet'].max()}")

# Si > 1, c'est déjà en %, sinon multiplier par 100
df_clean['taux_interet'] = df_clean['taux_interet'].apply(
    lambda x: x if x > 1 else x * 100 if pd.notna(x) else x
)

# Borner aux valeurs raisonnables (5% - 30%)
df_clean['taux_interet'] = df_clean['taux_interet'].clip(5, 30)

print(f"Après - Min: {df_clean['taux_interet'].min()}, Max: {df_clean['taux_interet'].max()}")

# 3. Conversion des dates
print("\n\nConversion des dates:")
df_clean['date_octroi'] = pd.to_datetime(df_clean['date_octroi'], errors='coerce')
df_clean['date_echeance'] = pd.to_datetime(df_clean['date_echeance'], errors='coerce')
df_clean['date_naissance'] = pd.to_datetime(df_clean['date_naissance'], errors='coerce')

# Corriger les dates impossibles (après aujourd'hui ou très anciennes)
today = pd.Timestamp.now()
df_clean.loc[df_clean['date_octroi'] > today, 'date_octroi'] = pd.NaT
df_clean.loc[df_clean['date_naissance'] > today - pd.Timedelta(days=18*365), 'date_naissance'] = pd.NaT

# 4. Correction des montants
print("\n\nCorrection des montants:")
# Montants négatifs -> valeur absolue
df_clean['montant_initial'] = df_clean['montant_initial'].abs()

# Solde > Montant -> plafonner au montant
df_clean['solde_restant'] = df_clean[['solde_restant', 'montant_initial']].min(axis=1)

# Solde négatif -> 0
df_clean['solde_restant'] = df_clean['solde_restant'].clip(lower=0)

# 5. Score dans les limites
df_clean['score_interne'] = df_clean['score_interne'].clip(300, 850)

# 6. Standardisation du secteur
df_clean['secteur_activite'] = df_clean['secteur_activite'].str.strip().str.title()
```

```
print("\nColonnes après nettoyage:")
print(df_clean.dtypes)
```

Exercice 2.3

Détectez et traitez les outliers.

Solution 2.3

```
print("TRAITEMENT DES OUTLIERS")
print("=" * 50)

def detect_outliers_iqr(df, column):
    """Détecte les outliers avec la méthode IQR"""
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    outliers = df[(df[column] < lower) | (df[column] > upper)]
    return outliers, lower, upper

# Variables à analyser
numeric_cols = ['montant_initial', 'solde_restant', 'revenu_mensuel',
                'anciennete_emploi', 'valeur_garantie']

outliers_summary = []

for col in numeric_cols:
    outliers, lower, upper = detect_outliers_iqr(df_clean, col)
    n_outliers = len(outliers)
    pct = n_outliers / len(df_clean) * 100

    outliers_summary.append({
        'Variable': col,
        'N_Outliers': n_outliers,
        'Pourcentage': f"{pct:.2f}%",
        'Borne_Inf': f"{lower:,.0f}",
        'Borne_Sup': f"{upper:,.0f}"
    })

print(pd.DataFrame(outliers_summary))

# Traitement: Capping (Winsorization)
print("\n\nApplication du capping:")

for col in numeric_cols:
    Q1 = df_clean[col].quantile(0.01)  # 1er percentile
    Q99 = df_clean[col].quantile(0.99)  # 99ème percentile
```

```
    before = df_clean[col].describe()
    df_clean[col] = df_clean[col].clip(Q1, Q99)
    after = df_clean[col].describe()

    print(f"\n{col}:")
    print(f"  Avant - Min: {before['min']:,.0f}, Max: {before['max']:,.0f}")
    print(f"  Après - Min: {after['min']:,.0f}, Max: {after['max']:,.0f}")

# Visualisation avant/après
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].boxplot(df['montant_initial'].dropna())
axes[0].set_title('Montant Initial - Avant')
axes[1].boxplot(df_clean['montant_initial'].dropna())
axes[1].set_title('Montant Initial - Après')
plt.savefig('outliers_comparison.png')
```

PARTIE 3: Feature Engineering

Exercice 3.1

Créez des variables dérivées pertinentes pour l'analyse de risque.

Solution 3.1

```
print("FEATURE ENGINEERING")
print("=" * 50)

# 1. Variables temporelles
df_clean['age'] = (today - df_clean['date_naissance']).dt.days // 365
df_clean['ancienneté_pret_mois'] = (today - df_clean['date_octroi']).dt.days // 30
df_clean['duree_pret_mois'] = (df_clean['date_echeance'] - df_clean['date_octroi']).dt.days // 30
df_clean['mois_restants'] = (df_clean['date_echeance'] - today).dt.days // 30
df_clean['mois_restants'] = df_clean['mois_restants'].clip(lower=0)

# 2. Ratios financiers
df_clean['DTI'] = df_clean['solde_restant'] / df_clean['revenu_mensuel'].replace(0, np.nan)
df_clean['DTI'] = df_clean['DTI'].clip(0, 10)  # Plafonner à 10x

df_clean['LTV'] = df_clean['montant_initial'] / df_clean['valeur_garantie'].replace(0, np.nan)
df_clean['LTV'] = df_clean['LTV'].fillna(999)  # 999 si pas de garantie
df_clean['LTV'] = df_clean['LTV'].clip(0, 999)

df_clean['pct_rembourse'] = 1 - (df_clean['solde_restant'] / df_clean['montant_initial'].replace(0, np.n
df_clean['pct_rembourse'] = df_clean['pct_rembourse'].clip(0, 1)

# 3. Indicateurs de risque
df_clean['is_npl'] = (df_clean['jours_retard'] > 90).astype(int)
df_clean['is_retard'] = (df_clean['jours_retard'] > 0).astype(int)

df_clean['bucket_retard'] = pd.cut(
    df_clean['jours_retard'],
    bins=[-1, 0, 30, 60, 90, 999999],
```

```
    labels=['Current', '1-30j', '31-60j', '61-90j', '90+j']
)

# 4. Catégorisation du score
df_clean['score_category'] = pd.cut(
    df_clean['score_interne'],
    bins=[0, 500, 600, 700, 850],
    labels=['Faible', 'Moyen', 'Bon', 'Excellent']
)

# 5. Catégorisation de l'âge
df_clean['age_group'] = pd.cut(
    df_clean['age'],
    bins=[0, 25, 35, 45, 55, 100],
    labels=['18-25', '26-35', '36-45', '46-55', '55+']
)

# 6. Taille du prêt
df_clean['taille_pret'] = pd.qcut(
    df_clean['montant_initial'],
    q=4,
    labels=['Petit', 'Moyen', 'Grand', 'Très Grand']
)

# 7. Variables comportementales (si historique disponible)
# Exemple: nb_credits_anterieurs -> catégorie
df_clean['experience_credit'] = pd.cut(
    df_clean['nb_credits_anterieurs'],
    bins=[-1, 0, 2, 5, 100],
    labels=['Nouveau', 'Peu expérimenté', 'Expérimenté', 'Très expérimenté']
)

print("\nNouvelles variables créées:")
new_cols = ['age', 'anciennete_pret_mois', 'duree_pret_mois', 'DTI', 'LTV',
            'pct_rembourse', 'is_npl', 'bucket_retard', 'score_category',
            'age_group', 'taille_pret', 'experience_credit']
print(df_clean[new_cols].head(10))

print("\nStatistiques des nouvelles variables:")
print(df_clean[['DTI', 'LTV', 'pct_rembourse', 'age', 'anciennete_pret_mois']].describe())
```

PARTIE 4: Analyse et Insights

Exercice 4.1

Identifiez les principaux insights du portefeuille.

Solution 4.1

```
print("INSIGHTS CLÉS DU PORTEFEUILLE")
print("=" * 50)
```

```python
# 1. Taux de NPL global
npl_rate = df_clean['is_npl'].mean() * 100
print(f"\n1. TAUX DE NPL GLOBAL: {npl_rate:.2f}%")

# 2. NPL par secteur
npl_by_sector = df_clean.groupby('secteur_activite').agg({
    'is_npl': 'mean',
    'solde_restant': 'sum',
    'id_pret': 'count'
}).round(4)
npl_by_sector.columns = ['Taux_NPL', 'Encours', 'Nb_Prets']
npl_by_sector = npl_by_sector.sort_values('Taux_NPL', ascending=False)
print("\n2. NPL PAR SECTEUR:")
print(npl_by_sector)

# 3. NPL par catégorie de score
npl_by_score = df_clean.groupby('score_category')['is_npl'].mean() * 100
print("\n3. NPL PAR CATÉGORIE DE SCORE:")
print(npl_by_score)

# 4. Corrélations avec le défaut
numeric_for_corr = ['montant_initial', 'score_interne', 'DTI', 'LTV',
                    'age', 'anciennete_emploi', 'revenu_mensuel']
correlations = df_clean[numeric_for_corr + ['is_npl']].corr()['is_npl'].drop('is_npl')
correlations = correlations.sort_values(key=abs, ascending=False)
print("\n4. CORRÉLATIONS AVEC LE DÉFAUT:")
print(correlations)

# 5. Profil type du client en défaut vs sain
print("\n5. PROFIL COMPARÉ:")
comparison = df_clean.groupby('is_npl')[numeric_for_corr].mean()
comparison.index = ['Sain', 'Défaut']
print(comparison.T)

# 6. Concentration du portefeuille
print("\n6. CONCENTRATION:")
top_10_clients = df_clean.groupby('id_client')['solde_restant'].sum().nlargest(10)
total_encours = df_clean['solde_restant'].sum()
print(f"   Top 10 clients: {top_10_clients.sum() / total_encours * 100:.1f}% de l'encours")

# Visualisation finale
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# NPL par secteur
npl_by_sector['Taux_NPL'].sort_values().plot(kind='barh', ax=axes[0,0])
axes[0,0].set_title('Taux de NPL par Secteur')
axes[0,0].set_xlabel('Taux de NPL')

# NPL par score
npl_by_score.plot(kind='bar', ax=axes[0,1], color=['red', 'orange', 'yellow', 'green'])
axes[0,1].set_title('Taux de NPL par Catégorie de Score')
axes[0,1].set_ylabel('Taux NPL (%)')

# Distribution buckets
```

```
df_clean['bucket_retard'].value_counts().plot(kind='pie', ax=axes[1,0], autopct='%1.1f%%')
axes[1,0].set_title('Répartition par Bucket de Retard')

# Corrélations
correlations.plot(kind='barh', ax=axes[1,1])
axes[1,1].set_title('Corrélation avec le Défaut')
axes[1,1].axvline(x=0, color='black', linestyle='--')

plt.tight_layout()
plt.savefig('insights_finaux.png')
plt.show()
```

PARTIE 5: Préparation pour le Scoring

Exercice 5.1

Préparez le dataset final pour la modélisation.

Solution 5.1

```
print("PRÉPARATION FINALE POUR MODÉLISATION")
print("=" * 50)

# 1. Sélection des features
features_numeriques = ['montant_initial', 'solde_restant', 'taux_interet',
                       'revenu_mensuel', 'anciennete_emploi', 'score_interne',
                       'age', 'DTI', 'LTV', 'pct_rembourse', 'nb_credits_anterieurs']

features_categorielles = ['sexe', 'secteur_activite', 'score_category',
                          'age_group', 'taille_pret', 'experience_credit', 'garantie']

target = 'is_npl'

# 2. Création du dataset final
df_final = df_clean[features_numeriques + features_categorielles + [target]].copy()

# 3. Encoding des variables catégorielles
from sklearn.preprocessing import LabelEncoder

for col in features_categorielles:
    le = LabelEncoder()
    df_final[col + '_encoded'] = le.fit_transform(df_final[col].astype(str))

# 4. Standardisation des variables numériques
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_final[features_numeriques] = scaler.fit_transform(df_final[features_numeriques])

# 5. Split train/test stratifié
from sklearn.model_selection import train_test_split
```

```python
X = df_final.drop(columns=[target] + features_categorielles)
y = df_final[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"\nDataset final:")
print(f"  Train: {len(X_train)} ({y_train.mean()*100:.1f}% NPL)")
print(f"  Test: {len(X_test)} ({y_test.mean()*100:.1f}% NPL)")
print(f"  Features: {X.shape[1]}")

# 6. Sauvegarde
df_final.to_csv('portefeuille_clean.csv', index=False)
print("\nDataset nettoyé sauvegardé: portefeuille_clean.csv")

# 7. Rapport de data quality final
print("\nRAPPORT QUALITÉ FINAL:")
print(f"  Lignes: {len(df_final)}")
print(f"  Colonnes: {len(df_final.columns)}")
print(f"  Valeurs manquantes: {df_final.isnull().sum().sum()}")
print(f"  Doublons: {df_final.duplicated().sum()}")
```

Livrables Attendus

1. **Code Python** complet et documenté
2. **Rapport EDA** avec visualisations
3. **Log des transformations** appliquées
4. **Dataset nettoyé** (CSV)
5. **Dictionnaire des variables** créées
6. **Synthèse des insights** clés

Critères d'Évaluation

Critère	Points
Exploration complète	20%
Qualité du nettoyage	25%
Pertinence du feature engineering	25%
Insights identifiés	20%
Code propre et documenté	10%

Checklist EDA

```
Chargement et aperçu des données
Analyse des valeurs manquantes
Détection des doublons
Analyse des distributions
```

Détection des anomalies
Standardisation des formats
Traitement des outliers
Création de nouvelles variables
Analyse des corrélations
Identification des insights
Préparation pour modélisation
Documentation du processus