



```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
import kagglehub # Added for Kaggle dataset download
warnings.filterwarnings('ignore')

# Machine Learning
from sklearn.model_selection import train_test_split, GridSearchCV, learn
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import (classification_report, confusion_matrix,
                             accuracy_score, precision_score, recall_score,
                             f1_score, roc_curve, auc, roc_auc_score)
from sklearn.utils import resample

# Statistiques
from scipy import stats
from scipy.stats import chi2_contingency

# Configuration des graphiques
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 10

print("✓ Toutes les bibliothèques importées avec succès!")
print("="*80)

```

✓ Toutes les bibliothèques importées avec succès!

```

print("\n" + "="*80)
print("ÉTAPE 1: CHARGEMENT ET COMPRÉHENSION DES DONNÉES")
print("="*80)

download_path = kagglehub.dataset_download("lihx1hx/give-me-some-credit")

print("Path to dataset files:", download_path)

# Load the training data from the downloaded path
df = pd.read_csv(os.path.join(download_path, 'cs-training.csv'))
data = df

print(f"\n📊 Dataset chargé: {data.shape[0]} lignes × {data.shape[1]} colo

# Suppression de la colonne index si elle existe
if 'Unnamed: 0' in data.columns:
    data = data.drop('Unnamed: 0', axis=1)

```

```

print("\n ✓ Colonne index supprimée")

# Affichage des premières lignes
print("\n 📄 Aperçu des données:")
print(data.head())

print("\n 📊 Informations sur le dataset:")
print(data.info())

print("\n 📈 Statistiques descriptives:")
print(data.describe())

```

## ÉTAPE 1: CHARGEMENT ET COMPRÉHENSION DES DONNÉES

Warning: Looks like you're using an outdated `kagglehub` version (install  
 Downloading from <https://www.kaggle.com/api/v1/datasets/download/lihxlhx/give-me-some-money>  
 100%|██████████| 5.16M/5.16M [00:00<00:00, 59.0MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/lihxlhx/give-me-some-money

📊 Dataset chargé: 150000 lignes × 12 colonnes  
 ✓ Colonne index supprimée

📄 Aperçu des données:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	\
0	1	0.766127	45	
1	0	0.957151	40	
2	0	0.658180	38	
3	0	0.233810	30	
4	0	0.907239	49	

	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	\
0	2	0.802982	9120.0	
1	0	0.121876	2600.0	
2	1	0.085113	3042.0	
3	0	0.036050	3300.0	
4	1	0.024926	63588.0	

	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	\
0	13	0	
1	4	0	
2	2	1	
3	5	0	
4	7	0	

	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	\
0	6	0	
1	0	0	
2	0	0	
3	0	0	
4	1	0	

	NumberOfDependents
0	2.0
1	1.0
2	0.0
3	0.0

```
3          0.0
4          0.0
```



Informations sur le dataset:

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 150000 entries, 0 to 149999
```

```
Data columns (total 11 columns):
```

#	Column	Non-Null Count	Dtype
0	SeriousDlqin2yrs	150000 non-null	int64
1	RevolvingUtilizationOfUnsecuredLines	150000 non-null	float64
2	age	150000 non-null	int64

```
print("\n" + "="*80)
print("ÉTAPE 2: ANALYSE DESCRIPTIVE ET VISUALISATIONS")
print("="*80)

# 3.1 Distribution de la variable cible
print("\n📊 Distribution de la variable cible (SeriousDlqin2yrs):")
target_counts = data['SeriousDlqin2yrs'].value_counts()
print(target_counts)
print(f"\nPourcentage de détresse: {target_counts[1]/len(data)*100:.2f}%")
print(f"Déséquilibre des classes: {target_counts[0]/target_counts[1]:.2f}")

# Visualisation 1: Distribution de la cible
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Graphique en barres
axes[0].bar(['Sain (0)', 'Détresse (1)'], target_counts.values,
            color=['#2ecc71', '#e74c3c'], alpha=0.7, edgecolor='black')
axes[0].set_ylabel('Nombre d\'observations')
axes[0].set_title('Distribution de la Variable Cible', fontsize=14, fontw
axes[0].grid(axis='y', alpha=0.3)
for i, v in enumerate(target_counts.values):
    axes[0].text(i, v + 500, str(v), ha='center', fontweight='bold')

# Graphique en camembert
colors = ['#2ecc71', '#e74c3c']
explode = (0.05, 0.05)
axes[1].pie(target_counts.values, labels=['Sain', 'Détresse'], autopct='%
            colors=colors, explode=explode, shadow=True, startangle=90)
axes[1].set_title('Proportion des Classes', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.savefig('01_distribution_cible.png', dpi=300, bbox_inches='tight')
plt.show()
print("✓ Graphique sauvegardé: 01_distribution_cible.png")
```

```
=====
ÉTAPE 2: ANALYSE DESCRIPTIVE ET VISUALISATIONS
=====
```



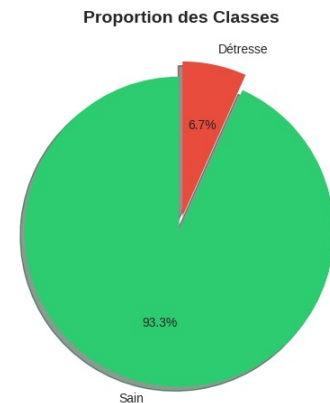
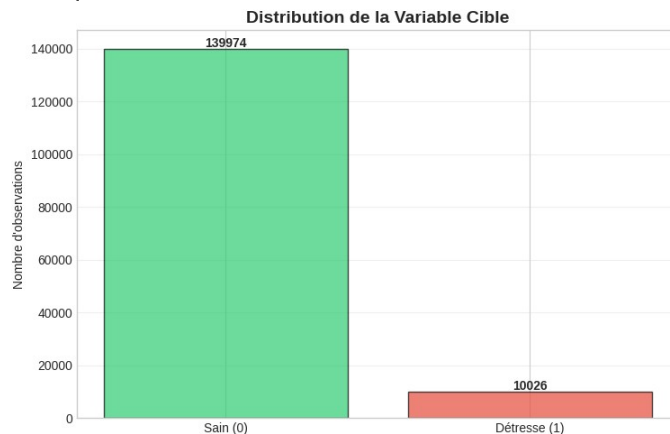
Distribution de la variable cible (SeriousDlqin2yrs):

```
SeriousDlqin2yrs
```

```
0    139974
```

1 10026  
Name: count, dtype: int64

Pourcentage de détresse: 6.68%  
Déséquilibre des classes: 13.96:1




✓ Graphique sauvegardé: 01\_distribution\_cible.png

```
# 3.2 Analyse des valeurs manquantes
print("\n🇫🇷 Analyse des valeurs manquantes:")
missing_data = data.isnull().sum()
missing_percent = (missing_data / len(data)) * 100
missing_df = pd.DataFrame({
    'Colonne': missing_data.index,
    'Valeurs Manquantes': missing_data.values,
    'Pourcentage': missing_percent.values
}).sort_values('Valeurs Manquantes', ascending=False)

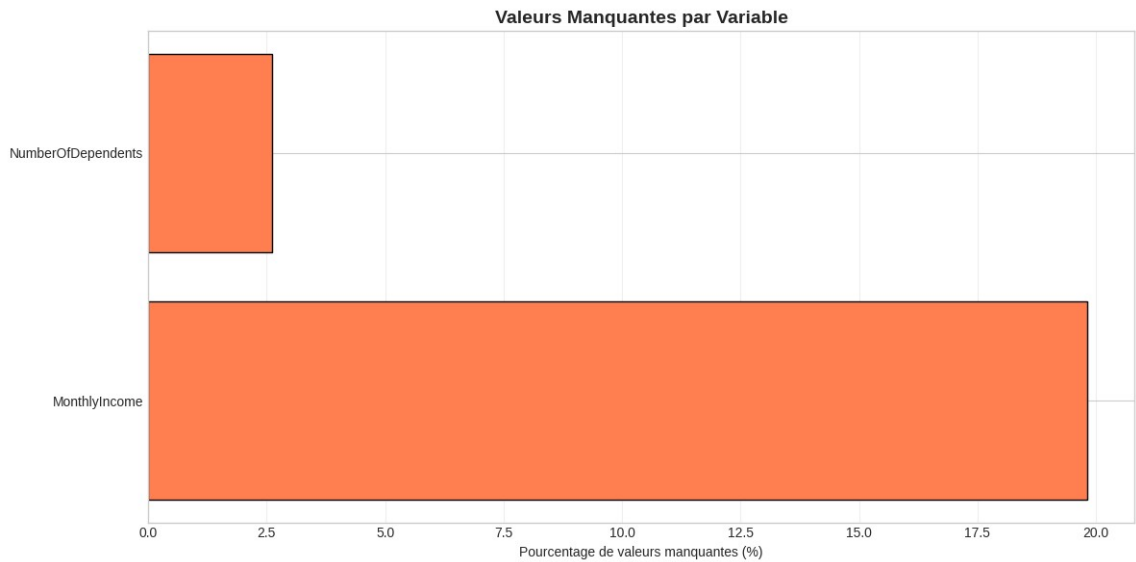
print(missing_df[missing_df['Valeurs Manquantes'] > 0])

# Visualisation 2: Valeurs manquantes
plt.figure(figsize=(12, 6))
missing_cols = missing_df[missing_df['Valeurs Manquantes'] > 0]
if len(missing_cols) > 0:
    plt.barh(missing_cols['Colonne'], missing_cols['Pourcentage'], color=
    plt.xlabel('Pourcentage de valeurs manquantes (%)')
    plt.title('Valeurs Manquantes par Variable', fontsize=14, fontweight=
    plt.grid(axis='x', alpha=0.3)
    plt.tight layout()
```

```
plt.savefig('02_valeurs_manquantes.png', dpi=300, bbox_inches='tight')
plt.show()
print("✓ Graphique sauvegardé: 02_valeurs_manquantes.png")
```


 Analyse des valeurs manquantes:

	Colonne	Valeurs Manquantes	Pourcentage
5	MonthlyIncome	29731	19.820667
10	NumberOfDependents	3924	2.616000



✓ Graphique sauvegardé: 02\_valeurs\_manquantes.png

### # 3.3 Distribution des variables numériques

```
print("\n Distribution des variables numériques:")
```

```
numeric_cols = data.select_dtypes(include=[np.number]).columns.tolist()
numeric_cols.remove('SeriousDlqin2yrs') # Exclure la cible
```

```
fig, axes = plt.subplots(4, 3, figsize=(18, 16))
axes = axes.ravel()
```

```
for idx, col in enumerate(numeric_cols):
    if idx < len(axes):
```

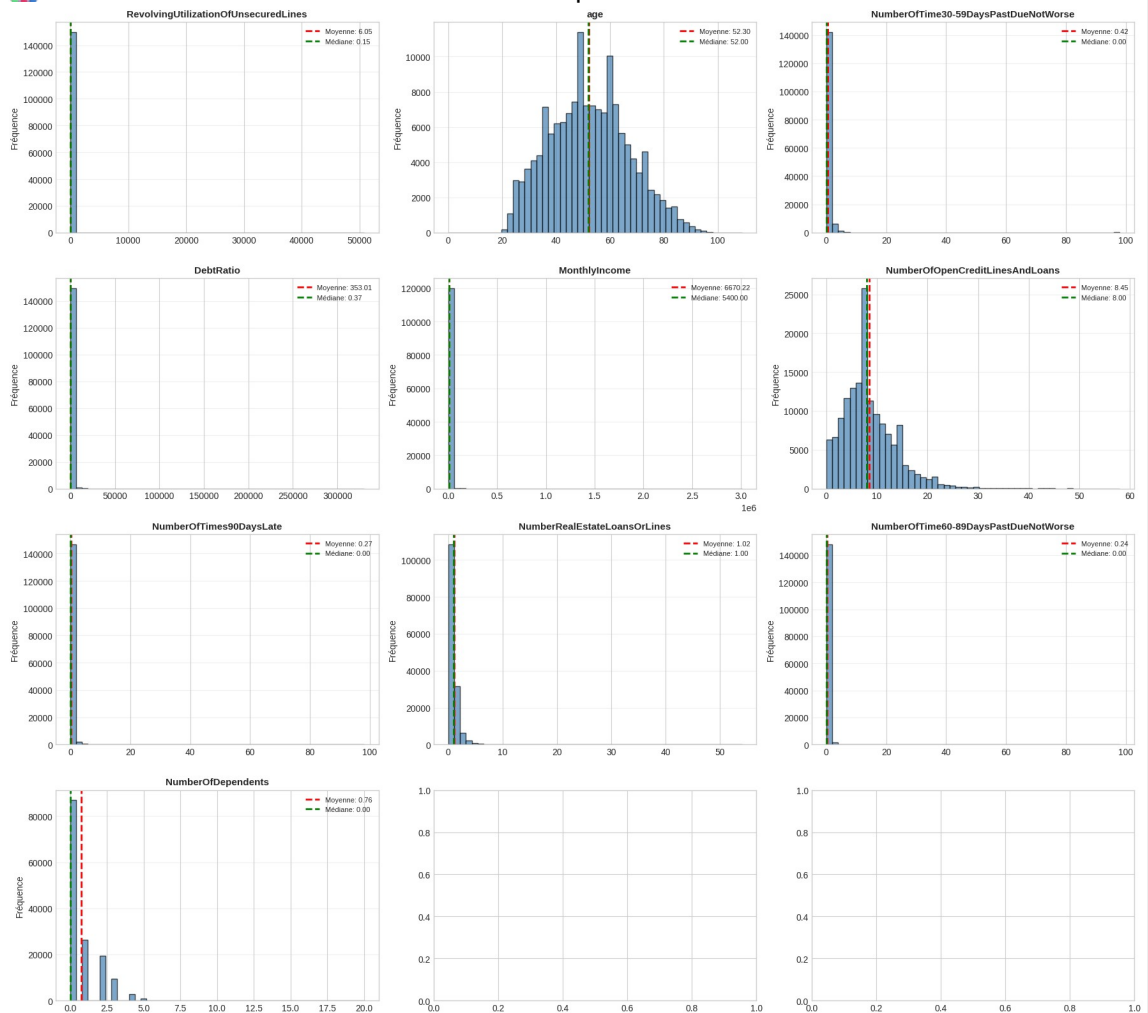
```
        # Histogramme des données KDE
```

```
# Histogramme avec KDE
axes[idx].hist(data[col].dropna(), bins=50, alpha=0.7, color='ste
axes[idx].set_title(f'{col}', fontsize=11, fontweight='bold')
axes[idx].set_ylabel('Fréquence')
axes[idx].grid(axis='y', alpha=0.3)

# Ajouter statistiques
mean_val = data[col].mean()
median_val = data[col].median()
axes[idx].axvline(mean_val, color='red', linestyle='--', linewidth
axes[idx].axvline(median_val, color='green', linestyle='--', line
axes[idx].legend(fontsize=8)

plt.tight_layout()
plt.savefig('03_distribution_variables.png', dpi=300, bbox_inches='tight'
plt.show()
print("✓ Graphique sauvegardé: 03_distribution_variables.png")
```

### Distribution des variables numériques:



✓ Graphique sauvegardé: 03\_distribution\_variables.png

```

# 3.4 Boxplots pour détecter les outliers
print("\n📊 Détection des valeurs aberrantes (Boxplots):")

fig, axes = plt.subplots(4, 3, figsize=(18, 16))
axes = axes.ravel()

for idx, col in enumerate(numeric_cols):
    if idx < len(axes):
        bp = axes[idx].boxplot(data[col].dropna(), vert=True, patch_artist=True)
        for patch in bp['boxes']:
            patch.set_facecolor('lightblue')
            patch.set_alpha(0.7)
        axes[idx].set_title(f'{col}', fontsize=11, fontweight='bold')
        axes[idx].set_ylabel('Valeur')
        axes[idx].grid(axis='y', alpha=0.3)

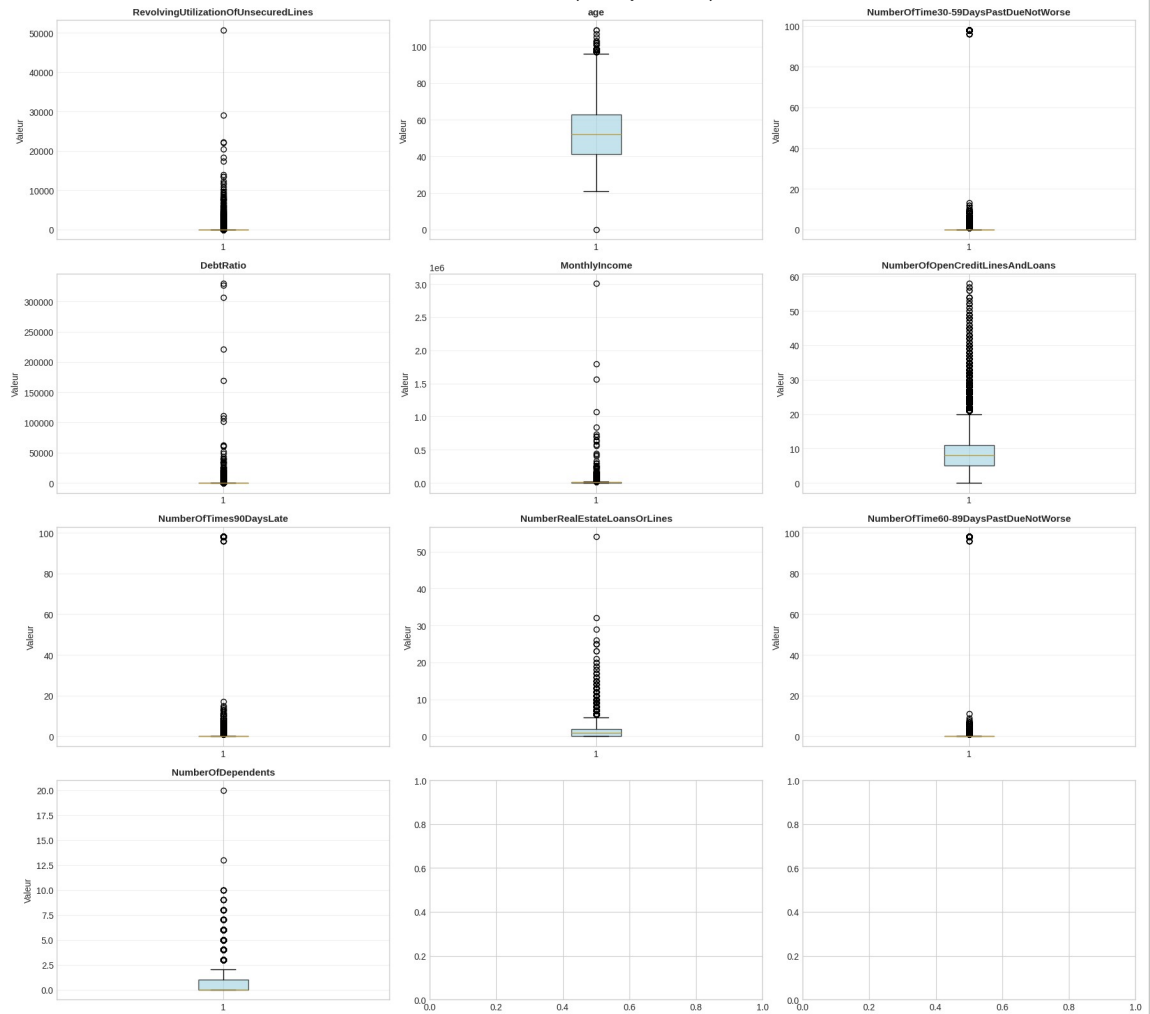
plt.tight_layout()
plt.savefig('04 boxplots outliers.png', dpi=300, bbox_inches='tight')

```



```
plt.show()
print("✓ Graphique sauvegardé: 04_boxplots_outliers.png")
```

### Détection des valeurs aberrantes (Boxplots):



✓ Graphique sauvegardé: 04\_boxplots\_outliers.png

# 3.5 Matrice de corrélation

```
print("\n📊 Matrice de corrélation:")
```

```
correlation_matrix = data.corr()
```

```
print("\nCorrélations avec la variable cible:")
```

```
target_corr = correlation_matrix['SeriousDlqin2yrs'].sort_values(ascending=False)
```

```
print(target_corr)
```

# Visualisation de la matrice de corrélation

```
plt.figure(figsize=(14, 12))
```

```
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
```

```
sns.heatmap(correlation_matrix, mask=mask, annot=True, fmt='.2f',  
            cmap='coolwarm', center=0, square=True, linewidths=1,  
            cbar_kws={"shrink": 0.8})
```

```
plt.title('Matrice de Corrélation des Variables', fontsize=16, fontweight='bold')
```

```
plt.tight_layout()
```

```
plt.savefig('05_matrice_correlation.png', dpi=300, bbox_inches='tight')
```

```
plt.show()
```

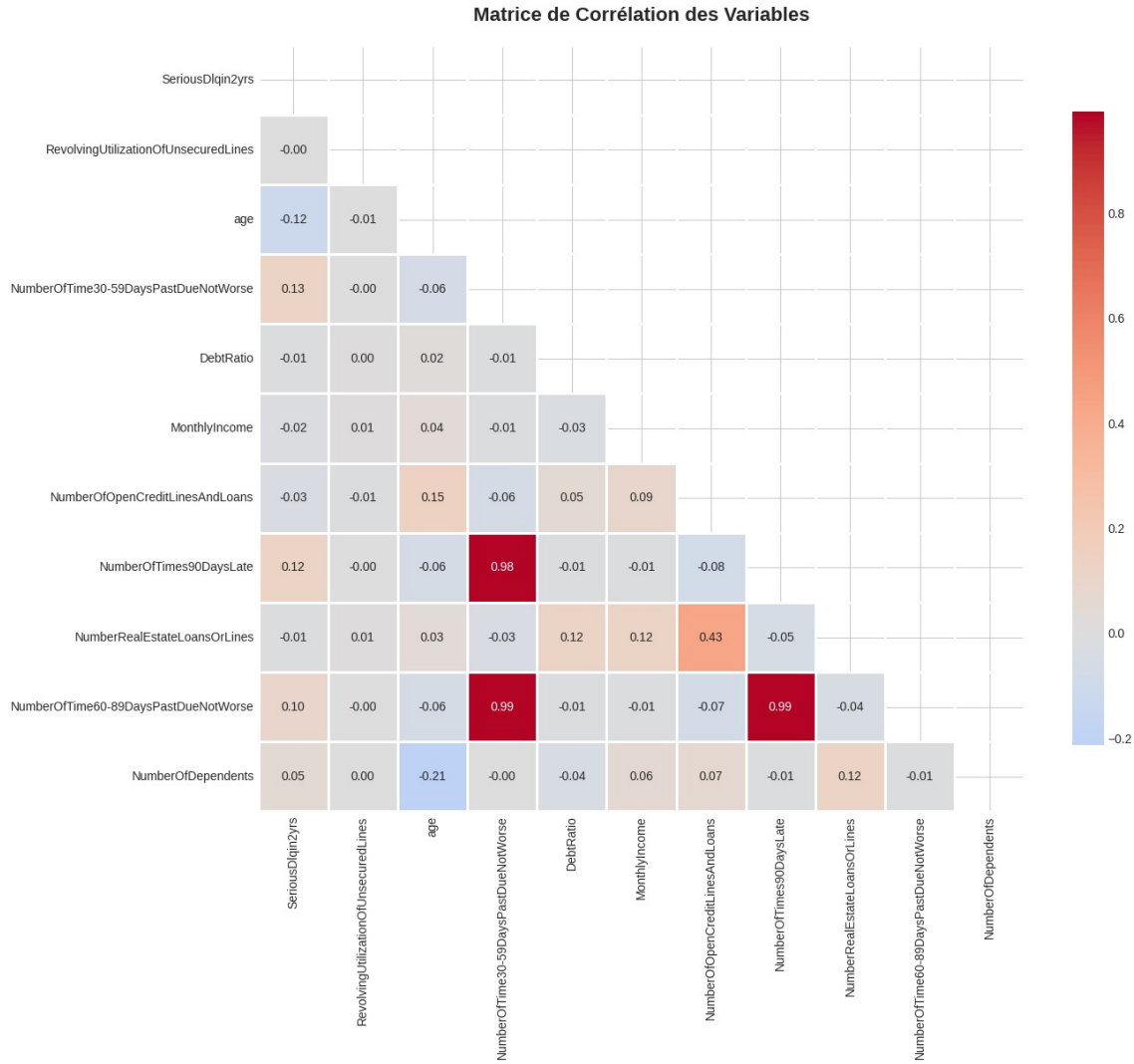
```
print("✓ Graphique sauvegardé: 05_matrice_correlation.png")
```

📊 Matrice de corrélation:

Corrélations avec la variable cible:

SeriousDlqin2yrs	1.000000
NumberOfTime30-59DaysPastDueNotWorse	0.125587
NumberOfTimes90DaysLate	0.117175
NumberOfTime60-89DaysPastDueNotWorse	0.102261
NumberOfDependents	0.046048
RevolvingUtilizationOfUnsecuredLines	-0.001802
NumberRealEstateLoansOrLines	-0.007038
DebtRatio	-0.007602
MonthlyIncome	-0.019746

NumberOfOpenCreditLinesAndLoans -0.029669  
age -0.115386  
Name: SeriousDlqin2yrs, dtype: float64



✓ Graphique sauvegardé: 05\_matrice\_correlation.png

```
# 3.6 Analyse bivariée: Variables vs Cible
print("\n📊 Analyse bivariée: Impact des variables sur la cible")

# Sélectionner quelques variables importantes
important_vars = ['age', 'MonthlyIncome', 'DebtRatio', 'NumberOfTimes90Da

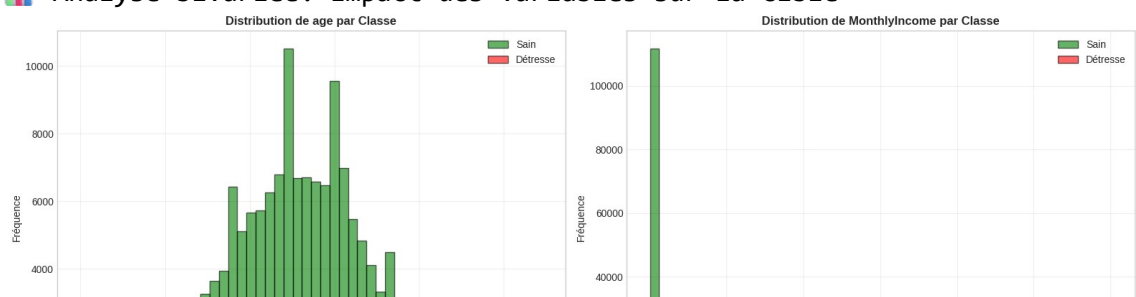
fig, axes = plt.subplots(2, 2, figsize=(16, 12))
axes = axes.ravel()

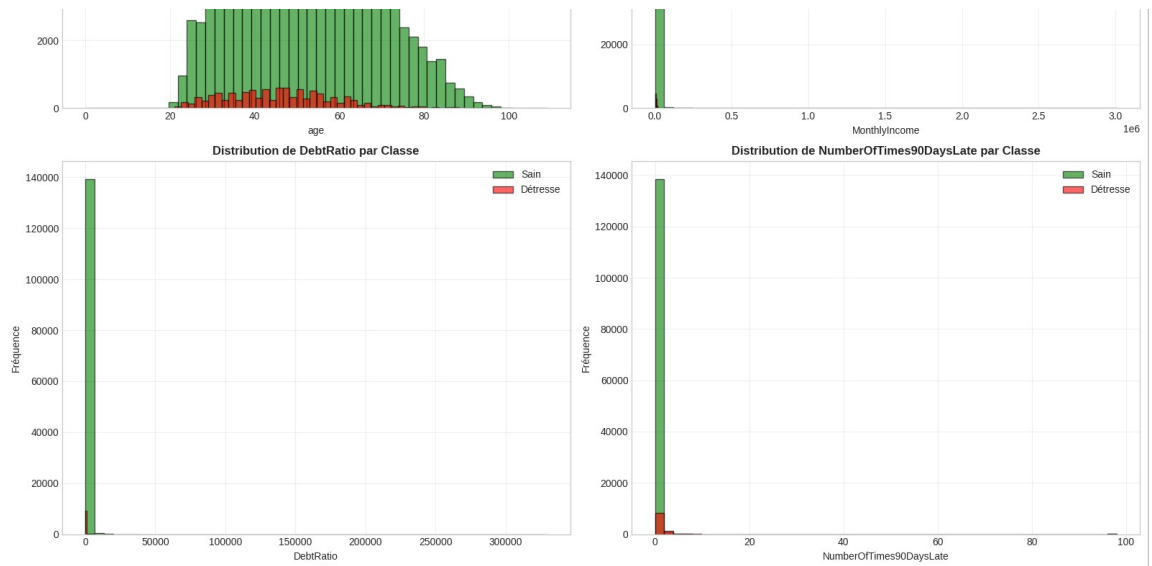
for idx, var in enumerate(important_vars):
    if var in data.columns:
        # Séparer par classe
        sain = data[data['SeriousDlqin2yrs'] == 0][var].dropna()
        detresse = data[data['SeriousDlqin2yrs'] == 1][var].dropna()

        # Histogrammes superposés
        axes[idx].hist(sain, bins=50, alpha=0.6, label='Sain', color='green')
        axes[idx].hist(detresse, bins=50, alpha=0.6, label='Détresse', color='red')
        axes[idx].set_xlabel(var)
        axes[idx].set_ylabel('Fréquence')
        axes[idx].set_title(f'Distribution de {var} par Classe', fontsize=12)
        axes[idx].legend()
        axes[idx].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('06_analyse_bivariee.png', dpi=300, bbox_inches='tight')
plt.show()
print("✓ Graphique sauvegardé: 06_analyse_bivariee.png")
```

### 📊 Analyse bivariée: Impact des variables sur la cible





✓ Graphique sauvegardé: 06\_analyse\_bivariee.png

```
print("\n" + "="*80)
print("ÉTAPE 3 : PRÉTRAITEMENT DES DONNÉES")
```

```

print("\n ÉTAPE 3: TRAITEMENT DES DONNÉES ")
print("="*80)

# Renommer les colonnes en français
print("\n 🔄 Renommage des colonnes en français...")
data.columns = [
    "DefautPaieement",          # SeriousDlqin2yrs
    "UtilisationCreditNonGaranti", # RevolvingUtilizationOfUnsecuredLin
    "Age",                      # age
    "Retard30_59Jours",         # NumberOfTime30-59DaysPastDueNotWor
    "RatioDette",              # DebtRatio
    "RevenuMensuel",           # MonthlyIncome
    "CreditsOuverts",          # NumberOfOpenCreditLinesAndLoans
    "Retard90Jours",            # NumberOfTimes90DaysLate
    "PretsImmobiliers",        # NumberRealEstateLoansOrLines
    "Retard60_89Jours",         # NumberOfTime60-89DaysPastDueNotWor
    "PersonnesACharge"          # NumberOfDependents
]
print("✓ Colonnes renommées")

# Gestion des valeurs manquantes
print("\n 🐛 Gestion des valeurs manquantes...")
print(f"Avant: {data.isnull().sum().sum()} valeurs manquantes")

data['RevenuMensuel'] = data['RevenuMensuel'].fillna(data['RevenuMensuel'])
data['PersonnesACharge'] = data['PersonnesACharge'].fillna(data['Personne'])

print(f"Après: {data.isnull().sum().sum()} valeurs manquantes")
print("✓ Valeurs manquantes traitées")

# Gestion des valeurs aberrantes
print("\n 🐛 Gestion des valeurs aberrantes...")
print(f"Taille initiale: {len(data)} lignes")

# Filtrer les valeurs aberrantes
data = data[data['UtilisationCreditNonGaranti'] <= 1]
print(f"Après filtrage UtilisationCreditNonGaranti: {len(data)} lignes")


data = data[(data['Age'] >= 18) & (data['Age'] <= 100)]
print(f"Après filtrage Age: {len(data)} lignes")

# Méthode IQR pour RatioDette
Q1 = data['RatioDette'].quantile(0.25)
Q3 = data['RatioDette'].quantile(0.75)
IQR = Q3 - Q1
borne_inf = Q1 - 1.5 * IQR
borne_sup = Q3 + 1.5 * IQR
data = data[(data['RatioDette'] >= borne_inf) & (data['RatioDette'] <= bo
print(f"Après filtrage RatioDette (IQR): {len(data)} lignes")


data = data[data['PersonnesACharge'] >= 0]
print(f"Taille finale: {len(data)} lignes")
print("✓ Valeurs aberrantes traitées")

```

### ÉTAPE 3: PRÉTRAITEMENT DES DONNÉES

 Renommage des colonnes en français...


✓ Colonnes renommées

 Gestion des valeurs manquantes...

Avant: 33655 valeurs manquantes

Après: 0 valeurs manquantes

✓ Valeurs manquantes traitées

 Gestion des valeurs aberrantes...

Taille initiale: 150000 lignes

Après filtrage UtilisationCreditNonGaranti: 146679 lignes

Après filtrage Age: 146665 lignes

Après filtrage RatioDette (IQR): 115972 lignes

Taille finale: 115972 lignes

✓ Valeurs aberrantes traitées

```
print("\n" + "="*80)
print("ÉTAPE 4: PARTITIONNEMENT DES DONNÉES")
print("="*80)

# Séparation features/target
X = data.drop('DefautPaieement', axis=1)
y = data['DefautPaieement']

print(f"\n📊 Features (X): {X.shape}")
print(f"📊 Target (y): {y.shape}")

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

print("\n✓ Normalisation appliquée (StandardScaler)")
print(f"Moyenne des features: {X_scaled.mean():.6f}")
print(f"Écart-type des features: {X_scaled.std():.6f}")

# Split Train/Test (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

print(f"\n📊 Ensemble d'entraînement: {len(X_train)} exemples ({len(X_train)/len(X_scaled)*100}%)")
print(f"📊 Ensemble de test: {len(X_test)} exemples ({len(X_test)/len(X_scaled)*100}%)")

print("\n📊 Distribution dans le train:")
print(y_train.value_counts())
print("\n📊 Distribution dans le test:")
print(y_test.value_counts())
```

### ÉTAPE 4: PARTITIONNEMENT DES DONNÉES

```
=====

📊 Features (X): (115972, 10)
📊 Target (y): (115972,)

✓ Normalisation appliquée (StandardScaler)
Moyenne des features: -0.000000
Écart-type des features: 1.000000

📊 Ensemble d'entraînement: 92777 exemples (80.0%)
📊 Ensemble de test: 23195 exemples (20.0%)

📊 Distribution dans le train:
DefaultPaieement
0      86977
1      5800
Name: count, dtype: int64

📊 Distribution dans le test:
DefaultPaieement
0      21745
1      1450
Name: count, dtype: int64
```

```
print("\n" + "="*80)
print("ÉTAPE 5: MODÉLISATION - NAIVE BAYES")
print("="*80)

# 6.1 Modèle de base
print("\n🤖 Entraînement du modèle Naive Bayes de base...")
nb_model_base = GaussianNB()
nb_model_base.fit(X_train, y_train)
y_pred_nb_base = nb_model_base.predict(X_test)

print("\n📊 Résultats Naive Bayes (modèle de base):")
print(f"Accuracy: {accuracy_score(y_test, y_pred_nb_base):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_nb_base, zero_division")
print(f"Recall: {recall_score(y_test, y_pred_nb_base):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred_nb_base):.4f}")
```

```
=====
ÉTAPE 5: MODÉLISATION - NAIVE BAYES
=====
```

```
🤖 Entraînement du modèle Naive Bayes de base...

📊 Résultats Naive Bayes (modèle de base):
Accuracy: 0.9352
Precision: 0.3197
Recall: 0.0324
F1-Score: 0.0589
```

```
# 6.2 Tuning des hyperparamètres
print("\n🔧 Tuning des hyperparamètres Naive Bayes...")
print("(Variation du paramètre var_smoothing)")
```



```

param_grid_nb = {
    'var_smoothing': np.logspace(-12, -6, 20)
}

grid_nb = GridSearchCV(GaussianNB(), param_grid_nb, cv=5,
                        scoring='f1', n_jobs=-1, verbose=1)
grid_nb.fit(X_train, y_train)

print(f"\n✓ Meilleurs paramètres: {grid_nb.best_params_}")
print(f"✓ Meilleur score F1 (CV): {grid_nb.best_score_:.4f}")

# Modèle optimisé
nb_model = grid_nb.best_estimator_
y_pred_nb = nb_model.predict(X_test)

print("\n📊 Résultats Naive Bayes (après tuning):")
print(classification_report(y_test, y_pred_nb))

```

🔧 Tuning des hyperparamètres Naive Bayes...  
 (Variation du paramètre var\_smoothing)  
 Fitting 5 folds for each of 20 candidates, totalling 100 fits

✓ Meilleurs paramètres: {'var\_smoothing': np.float64(1e-12)}  
 ✓ Meilleur score F1 (CV): 0.0613

📊 Résultats Naive Bayes (après tuning):

	precision	recall	f1-score	support
0	0.94	1.00	0.97	21745
1	0.32	0.03	0.06	1450
accuracy			0.94	23195
macro avg	0.63	0.51	0.51	23195
weighted avg	0.90	0.94	0.91	23195

```

print("\n" + "="*80)
print("ÉTAPE 6: MODÉLISATION - R-LEARNING")
print("="*80)

# 7.1 Préparation: Oversampling
print("\n🔄 Oversampling de la classe minoritaire...")

df_train = pd.DataFrame(X_train, columns=X.columns)
df_train['DefautPaieement'] = y_train.values

df_sain = df_train[df_train['DefautPaieement'] == 0]
df_detresse = df_train[df_train['DefautPaieement'] == 1]

print(f"Classe saine: {len(df_sain)}")
print(f"Classe détresse (avant): {len(df_detresse)}")

df_detresse_upsampled = resample(df_detresse, replace=True,

```

```

n_samples=len(df_sain)//2, random_state


print(f"Classe détresse (après): {len(df_detresse_upsampled)}")

df_balanced = pd.concat([df_sain, df_detresse_upsampled]).sample(frac=1,
X_train_balanced = df_balanced.drop('DefautPaieement', axis=1).values
y_train_balanced = df_balanced['DefautPaieement'].values

print(f"✓ Dataset équilibré: {len(X_train_balanced)} exemples")

```

## ÉTAPE 6: MODÉLISATION - R-LEARNING

 Oversampling de la classe minoritaire...

Classe saine: 86977

Classe détresse (avant): 5800

Classe détresse (après): 43488

✓ Dataset équilibré: 130465 exemples

### # 7.2 Implémentation de l'agent R-Learning

```
class RLearningAgent:
```

```
    """Agent de Reinforcement Learning (R-Learning)"""
```

```
    def __init__(self, alpha=0.01, beta=0.05, n_features=10):
```

```
        self.rho = 0.0
```

```
        self.alpha = alpha
```

```
        self.beta = beta
```

```
        self.q0 = SGDRegressor(loss='huber', learning_rate='adaptive', et
```

```
        self.q1 = SGDRegressor(loss='huber', learning_rate='adaptive', et
```

```
        dummy_state = np.zeros((1, n_features))
```

```
        self.q0.partial_fit(dummy_state, [0])
```

```
        self.q1.partial_fit(dummy_state, [0])
```

```
    def get_q_values(self, state):
```

```
        return np.array([self.q0.predict([state])[0], self.q1.predict([st
```

```
    def act(self, state, epsilon):
```

```
        if np.random.rand() < epsilon:
```

```
            return np.random.choice([0, 1])
```

```
        return np.argmax(self.get_q_values(state))
```

```
    def learn(self, s, a, r, s_next):
```

```
        q_next = self.get_q_values(s_next)
```

```
        max_q_next = np.max(q_next)
```

```
        target_q = r - self.rho + max_q_next
```

```
        model = self.q0 if a == 0 else self.q1
```

```
        model.partial_fit([s], [target_q])
```

```
        q_now = self.get_q_values(s)
```

```
        if a == np.argmax(q_now):
```

```

        diff = (r + max_q_next - np.max(q_now)) - self.rho
        self.rho += self.beta * diff

print("\n✓ Classe RLearningAgent définie")

```

✓ Classe RLearningAgent définie

```

# 7.3 Entraînement
print("\n🤖 Entraînement de l'agent R-Learning...")

agent = RLearningAgent(alpha=0.01, beta=0.05, n_features=X_train.shape[1])
epsilon = 1.0
rho_history = []

print(f"Nombre d'exemples: {len(X_train_balanced) - 1}")

for i in range(len(X_train_balanced) - 1):
    s = X_train_balanced[i]
    label = y_train_balanced[i]
    s_next = X_train_balanced[i + 1]

    action = agent.act(s, epsilon)

    if action == label:
        reward = 10
    else:
        reward = -50 if (label == 1 and action == 0) else -15

    agent.learn(s, action, reward, s_next)

    if i % 100 == 0:
        rho_history.append(agent.rho)

    epsilon *= 0.99995

    if (i + 1) % 10000 == 0:
        print(f" Progression: {i+1}/{len(X_train_balanced)-1} - Rho: {ag

print(f"\n✓ Entraînement terminé - Rho final: {agent.rho:.4f}")

```

```

🤖 Entraînement de l'agent R-Learning...
Nombre d'exemples: 130464
Progression: 10000/130464 - Rho: 0.5479
Progression: 20000/130464 - Rho: -3.9828
Progression: 30000/130464 - Rho: -2.8561
Progression: 40000/130464 - Rho: 2.2685
Progression: 50000/130464 - Rho: -5.3335
Progression: 60000/130464 - Rho: 2.4018
Progression: 70000/130464 - Rho: 2.2307
Progression: 80000/130464 - Rho: -1.5083
Progression: 90000/130464 - Rho: 0.2490
Progression: 100000/130464 - Rho: 1.7567
Progression: 110000/130464 - Rho: 0.8578

```

Progression: 120000/130464 - Rho: -0.2556  
Progression: 130000/130464 - Rho: -5.5772

✓ Entraînement terminé - Rho final: 4.1726

#### # 7.4 Prédications

```
print("\n🎮 Génération des prédictions...")

y_pred_rl = []
custom_threshold = 0.75

for i in range(len(X_test)):
    state = X_test[i]
    q_vals = agent.get_q_values(state)

    m = np.max(q_vals)
    exp_q = np.exp(q_vals - m)
    probs = exp_q / exp_q.sum()

    action = 1 if probs[1] > custom_threshold else 0
    y_pred_rl.append(action)

print("\n📊 Résultats R-Learning:")
print(classification_report(y_test, y_pred_rl))
```

🎮 Génération des prédictions...

📊 Résultats R-Learning:

	precision	recall	f1-score	support
0	0.97	0.78	0.87	21745
1	0.16	0.63	0.26	1450
accuracy			0.77	23195
macro avg	0.57	0.71	0.56	23195
weighted avg	0.92	0.77	0.83	23195

```
print("\n" + "="*80)
print("ÉTAPE 7: ANALYSE BIAIS / VARIANCE")
print("="*80)
```

#### # 8.1 Learning Curves pour Naive Bayes

```
print("\n📈 Génération des courbes d'apprentissage (Naive Bayes)...")
```

```
train_sizes, train_scores_nb, val_scores_nb = learning_curve(
    nb_model, X_train, y_train, cv=5, n_jobs=-1,
    train_sizes=np.linspace(0.1, 1.0, 10), scoring='f1'
)
```

```
train_mean_nb = np.mean(train_scores_nb, axis=1)
train_std_nb = np.std(train_scores_nb, axis=1)
val_mean_nb = np.mean(val_scores_nb, axis=1)
val_std_nb = np.std(val_scores_nb, axis=1)
```

```

val_std_nb = np.std(val_scores_nb, axis=1)

# Visualisation
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(train_sizes, train_mean_nb, 'o-', color='blue', label='Score Tra
plt.fill_between(train_sizes, train_mean_nb - train_std_nb,
                 train_mean_nb + train_std_nb, alpha=0.1, color='blue')
plt.plot(train_sizes, val_mean_nb, 'o-', color='red', label='Score Valida
plt.fill_between(train_sizes, val_mean_nb - val_std_nb,
                 val_mean_nb + val_std_nb, alpha=0.1, color='red')
plt.xlabel('Taille de l\'ensemble d\'entraînement')
plt.ylabel('Score F1')
plt.title('Courbe d\'Apprentissage - Naive Bayes', fontsize=14, fontweigh
plt.legend(loc='best')
plt.grid(alpha=0.3)

# Analyse Biaisi/Variance
gap_nb = train_mean_nb[-1] - val_mean_nb[-1]
print(f"\n📊 Analyse Naive Bayes:")
print(f"   Score Train final: {train_mean_nb[-1]:.4f}")
print(f"   Score Validation final: {val_mean_nb[-1]:.4f}")
print(f"   Écart (Gap): {gap_nb:.4f}")

if gap_nb > 0.1:
    print("   ⚠️ OVERFITTING détecté (variance élevée)")
elif val_mean_nb[-1] < 0.5:
    print("   ⚠️ UNDERFITTING détecté (biais élevé)")
else:
    print("   ✓ Bon équilibre biais/variance")

```

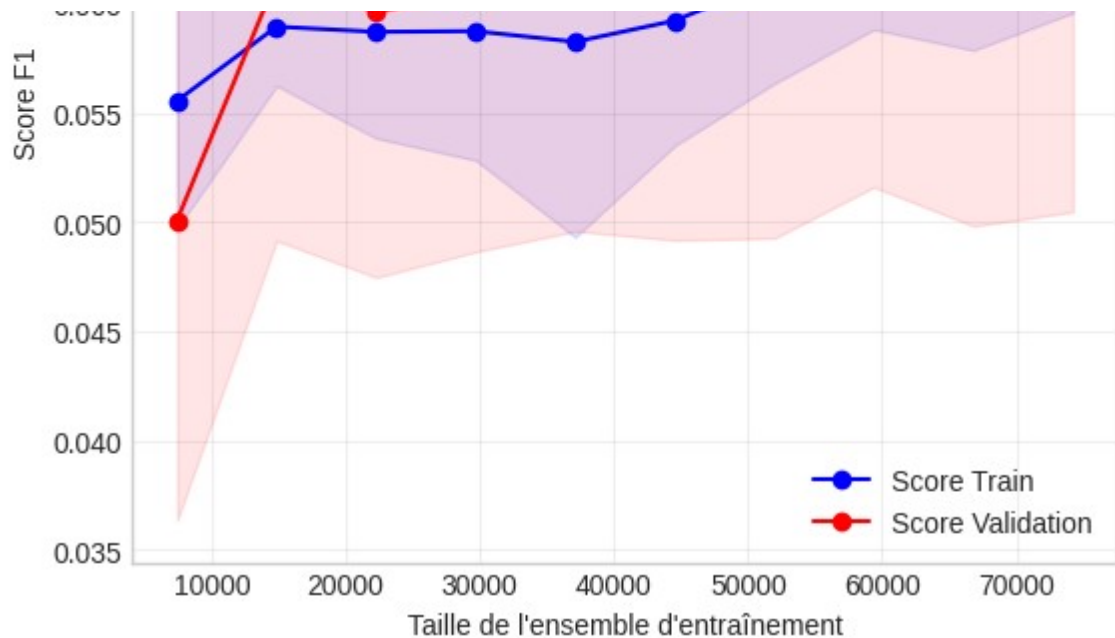
## ÉTAPE 7: ANALYSE BIAIS / VARIANCE

📊 Génération des courbes d'apprentissage (Naive Bayes)...

📊 Analyse Naive Bayes:  
 Score Train final: 0.0632  
 Score Validation final: 0.0613  
 Écart (Gap): 0.0019  
 ⚠️ UNDERFITTING détecté (biais élevé)

**Courbe d'Apprentissage - Naive Bayes**

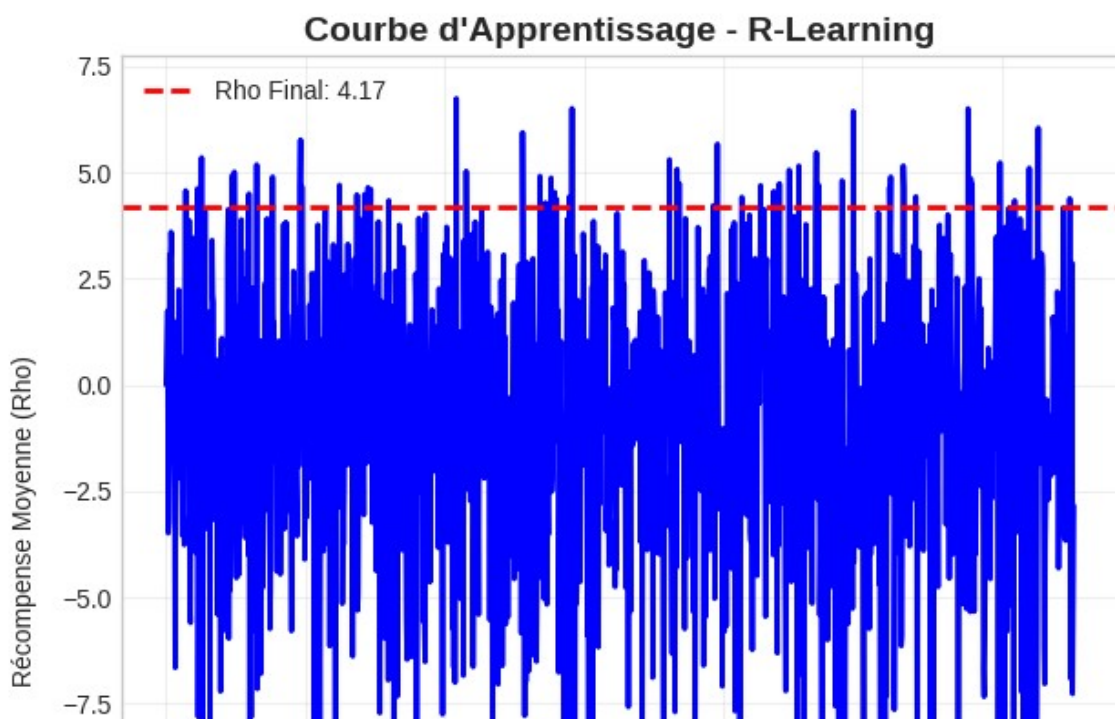


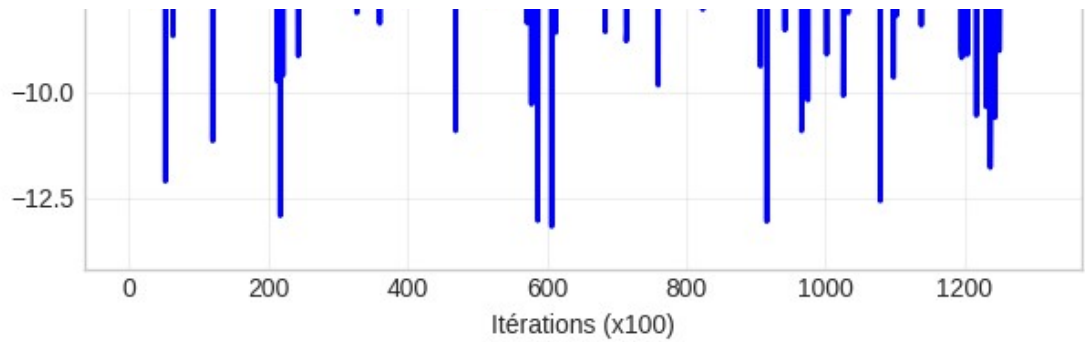


## # 8.2 Courbe d'apprentissage R-Learning

```
plt.subplot(1, 2, 2)
plt.plot(rho_history, color='blue', linewidth=2)
plt.axhline(y=agent.rho, color='red', linestyle='--', linewidth=2,
            label=f'Rho Final: {agent.rho:.2f}')
plt.xlabel('Itérations (x100)')
plt.ylabel('Récompense Moyenne (Rho)')
plt.title('Courbe d\'Apprentissage - R-Learning', fontsize=14, fontweight
plt.legend()
plt.grid(alpha=0.3)

plt.tight_layout()
plt.savefig('07_learning_curves.png', dpi=300, bbox_inches='tight')
plt.show()
print("✓ Graphique sauvegardé: 07_learning_curves.png")
```





✓ Graphique sauvegardé: 07\_learning\_curves.png

### # 8.3 Analyse détaillée Biais/Variance

```
print("\n" + "="*80)
print("ANALYSE DÉTAILLÉE BIAIS / VARIANCE")
print("="*80)

# Calculer les métriques sur train et test pour les deux modèles
print("\n🇫🇷 NAIVE BAYES:")
y_pred_nb_train = nb_model.predict(X_train)
train_acc_nb = accuracy_score(y_train, y_pred_nb_train)
test_acc_nb = accuracy_score(y_test, y_pred_nb)
train_f1_nb = f1_score(y_train, y_pred_nb_train)
test_f1_nb = f1_score(y_test, y_pred_nb)

print(f"  Accuracy Train: {train_acc_nb:.4f}")
print(f"  Accuracy Test:  {test_acc_nb:.4f}")
print(f"  F1-Score Train: {train_f1_nb:.4f}")
print(f"  F1-Score Test:  {test_f1_nb:.4f}")
print(f"  Écart Accuracy: {abs(train_acc_nb - test_acc_nb):.4f}")
print(f"  Écart F1-Score: {abs(train_f1_nb - test_f1_nb):.4f}")

if abs(train_acc_nb - test_acc_nb) > 0.05:
    print("  ⚠️  Variance élevée → Risque d'OVERFITTING")
else:
    print("  ✓ Variance acceptable")

if test_f1_nb < 0.3:
    print("  ⚠️  Biais élevé → UNDERFITTING sur classe minoritaire")
else:
    print("  ✓ Biais acceptable")

print("\n🇫🇷 R-LEARNING:")
test_acc_rl = accuracy_score(y_test, y_pred_rl)
test_f1_rl = f1_score(y_test, y_pred_rl)

print(f"  Accuracy Test:  {test_acc_rl:.4f}")
print(f"  F1-Score Test:  {test_f1_rl:.4f}")
print(f"  Rho convergé:   {agent.rho:.4f}")

if agent.rho < -10:
    print("  ⚠️  Rho très négatif → Difficulté d'apprentissage")
```

```

        print(" ⚠ Rho très négatif → Difficulté d'apprentissage ")
    else:
        print(" ✓ Convergence acceptable")

```

## =====

### ANALYSE DÉTAILLÉE BIAIS / VARIANCE

## =====

#### 🇫🇷 NAIVE BAYES:

```

Accuracy Train: 0.9357
Accuracy Test:  0.9352
F1-Score Train: 0.0628
F1-Score Test:  0.0589
Écart Accuracy: 0.0005
Écart F1-Score: 0.0040
✓ Variance acceptable
⚠ Biais élevé → UNDERFITTING sur classe minoritaire

```

#### 🇫🇷 R-LEARNING:

```

Accuracy Test:  0.7749
F1-Score Test:  0.2593
Rho convergé:   4.1726
✓ Convergence acceptable

```

```

print("\n" + "="*80)
print("ÉTAPE 8: ÉVALUATION ET COMPARAISON")
print("="*80)

# 9.1 Matrices de confusion
cm_nb = confusion_matrix(y_test, y_pred_nb)
cm_rl = confusion_matrix(y_test, y_pred_rl)

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Naive Bayes
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues', ax=axes[0],
            xticklabels=['Sain', 'Détresse'], yticklabels=['Sain', 'Détresse'],
            cbar_kws={'label': 'Nombre'})
axes[0].set_title('Matrice de Confusion - Naive Bayes', fontsize=14, fontweight='bold')
axes[0].set_ylabel('Vraie Classe')
axes[0].set_xlabel('Classe Prédite')

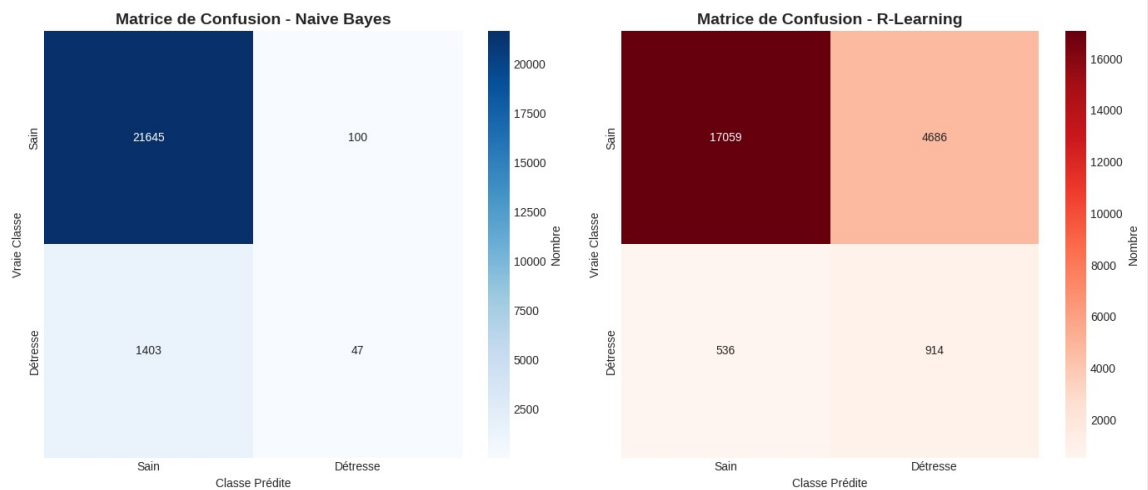
# R-Learning
sns.heatmap(cm_rl, annot=True, fmt='d', cmap='Reds', ax=axes[1],
            xticklabels=['Sain', 'Détresse'], yticklabels=['Sain', 'Détresse'],
            cbar_kws={'label': 'Nombre'})
axes[1].set_title('Matrice de Confusion - R-Learning', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Vraie Classe')
axes[1].set_xlabel('Classe Prédite')

plt.tight_layout()
plt.savefig('08_matrices_confusion.png', dpi=300, bbox_inches='tight')
plt.show()
print("✓ Graphique sauvegardé: 08_matrices_confusion.png")

```



## ÉTAPE 8: ÉVALUATION ET COMPARAISON



✓ Graphique sauvegardé: 08\_matrices\_confusion.png

```
# 9.2 Comparaison des métriques
metrics_comparison = pd.DataFrame({
    'Naive Bayes': {
        'Accuracy': accuracy_score(y_test, y_pred_nb),
        'Precision': precision_score(y_test, y_pred_nb, zero_division=0),
        'Recall': recall_score(y_test, y_pred_nb),
        'F1-Score': f1_score(y_test, y_pred_nb)
    },
    'R-Learning': {
        'Accuracy': accuracy_score(y_test, y_pred_rl),
        'Precision': precision_score(y_test, y_pred_rl, zero_division=0),
        'Recall': recall_score(y_test, y_pred_rl),
        'F1-Score': f1_score(y_test, y_pred_rl)
    }
})
```

print("\n 📊 TABLEAU COMPARATIF DES PERFORMANCES :\n")

```

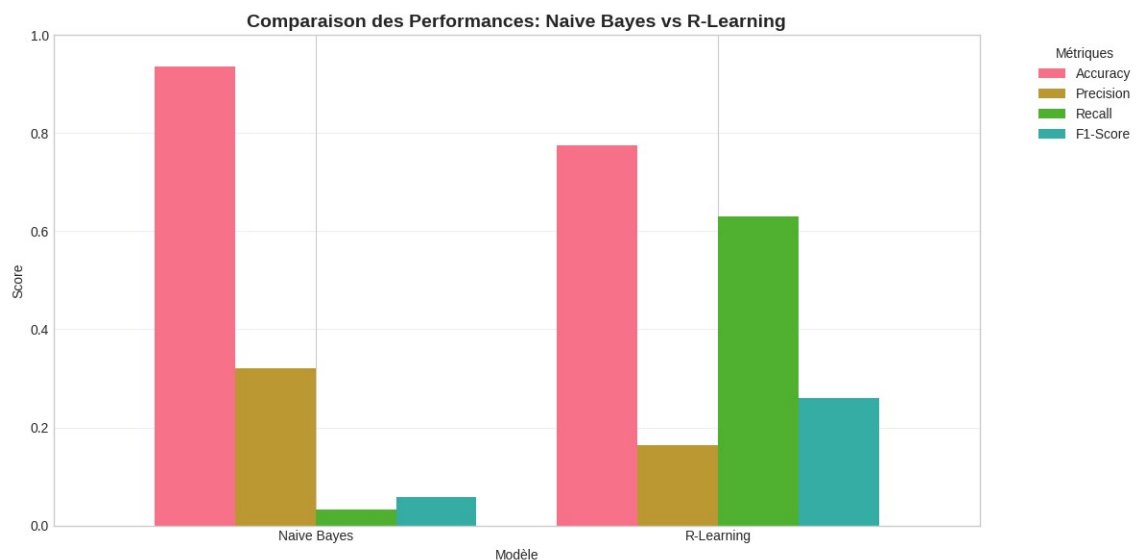
print("\n🇫🇷 TABLEAU COMPARATIF DES PERFORMANCES:")
print("="*60)
print(metrics_comparison.round(4))
print("="*60)

# Visualisation
metrics_comparison.T.plot(kind='bar', figsize=(12, 6), rot=0, width=0.8)
plt.title('Comparaison des Performances: Naive Bayes vs R-Learning',
          fontsize=14, fontweight='bold')
plt.ylabel('Score')
plt.xlabel('Modèle')
plt.legend(title='Métriques', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y', alpha=0.3)
plt.ylim(0, 1)
plt.tight_layout()
plt.savefig('09_comparaison_metriques.png', dpi=300, bbox_inches='tight')
plt.show()
print("✓ Graphique sauvegardé: 09_comparaison_metriques.png")

```

### 🇫🇷 TABLEAU COMPARATIF DES PERFORMANCES:

	Naive Bayes	R-Learning
Accuracy	0.9352	0.7749
Precision	0.3197	0.1632
Recall	0.0324	0.6303
F1-Score	0.0589	0.2593



✓ Graphique sauvegardé: 09\_comparaison\_metriques.png

### # 9.3 Courbes ROC

```
print("\n📊 Génération des courbes ROC...")
```

```
# Probabilités pour Naive Bayes
```

```
y_proba_nb = nb_model.predict_proba(X_test)[: , 1]
fpr_nb, tpr_nb, _ = roc_curve(y_test, y_proba_nb)
roc_auc_nb = auc(fpr_nb, tpr_nb)
```

```
# Pour R-Learning, utiliser les probabilités softmax
```

```
y_proba_rl = []
for i in range(len(X_test)):
    state = X_test[i]
    q_vals = agent.get_q_values(state)
    m = np.max(q_vals)
    exp_q = np.exp(q_vals - m)
    probs = exp_q / exp_q.sum()
    y_proba_rl.append(probs[1])
```

```
fpr_rl, tpr_rl, _ = roc_curve(y_test, y_proba_rl)
roc_auc_rl = auc(fpr_rl, tpr_rl)
```

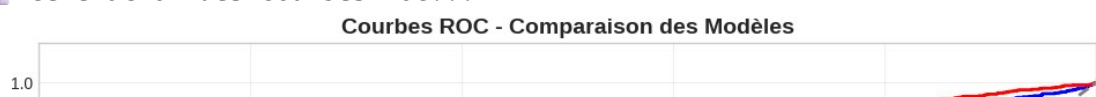
```
# Visualisation
```

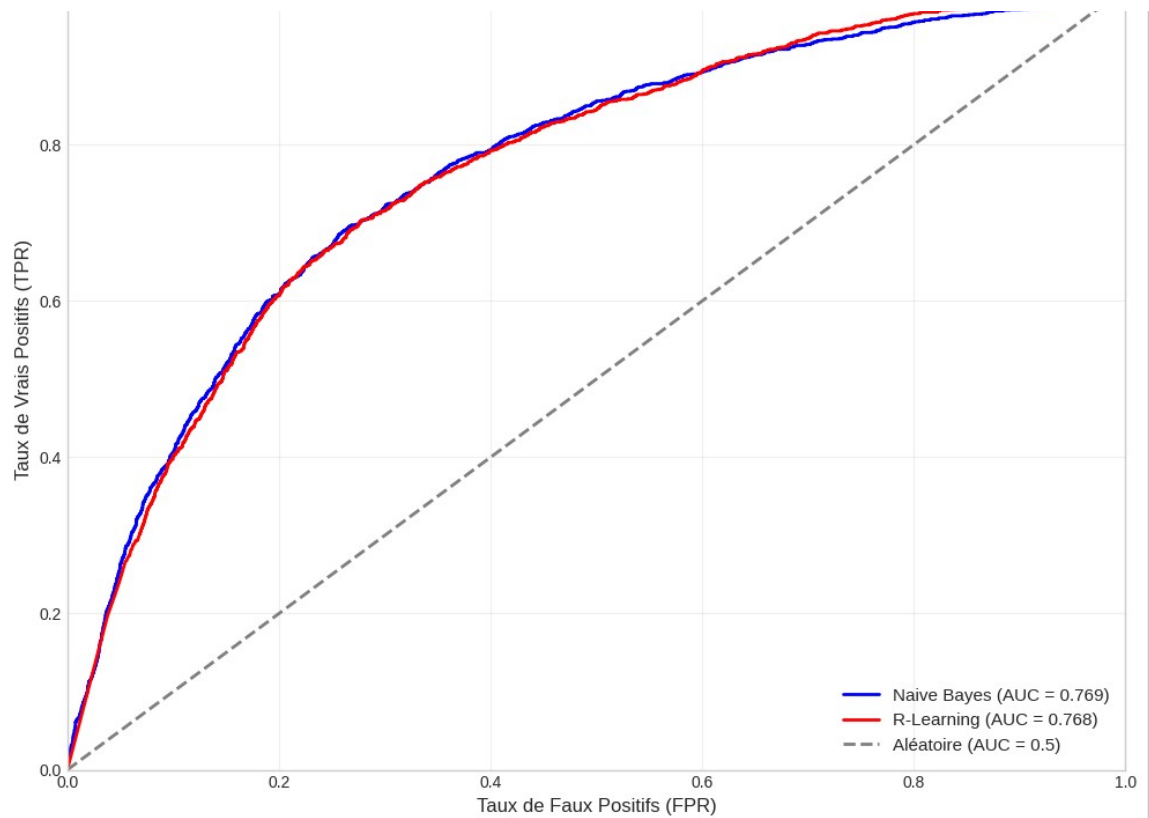
```
plt.figure(figsize=(10, 8))
plt.plot(fpr_nb, tpr_nb, color='blue', lw=2,
         label=f'Naive Bayes (AUC = {roc_auc_nb:.3f})')
plt.plot(fpr_rl, tpr_rl, color='red', lw=2,
         label=f'R-Learning (AUC = {roc_auc_rl:.3f})')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--', label='Aléat')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taux de Faux Positifs (FPR)', fontsize=12)
plt.ylabel('Taux de Vrais Positifs (TPR)', fontsize=12)
plt.title('Courbes ROC - Comparaison des Modèles', fontsize=14, fontweight='bold')
plt.legend(loc='lower right', fontsize=11)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.savefig('10_courbes_roc.png', dpi=300, bbox_inches='tight')
plt.show()
print("✓ Graphique sauvegardé: 10_courbes_roc.png")
```

```
print(f"\n📊 AUC Naive Bayes: {roc_auc_nb:.4f}")
```


```
print(f"📊 AUC R-Learning: {roc_auc_rl:.4f}")
```


📊 Génération des courbes ROC...





✓ Graphique sauvegardé: 10\_courbes\_roc.png

 AUC Naive Bayes: 0.7694

 AUC R-Learning: 0.7684

```
print("\n" + "="*80)
print("ÉTAPE 9: CONCLUSIONS ET RECOMMANDATIONS")
print("="*80)
```

```
print("""
```

```
CONCLUSIONS FINALES
```

## 1. ANALYSE DU PROBLEME

- ✓ Dataset déséquilibré (93% sains vs 7% détresse)
- ✓ Problème de classification binaire
- ✓ Objectif: Minimiser les faux négatifs (coût élevé)

## 2. NAIVE BAYES

Points forts:

- ✓ Très rapide à entraîner
- ✓ Bonne accuracy globale (~94%)
- ✓ Modèle simple et interprétable
- ✓ Faible variance (pas d'overfitting)

Points faibles:

- X Très faible recall sur classe minoritaire (~3%)
- X Biais élevé sur la détection de détresse
- X UNDERFITTING sur la classe 1
- X Assume l'indépendance des features

Diagnostic Biais/Variance:

- BIAIS ÉLEVÉ (Underfitting)
- Le modèle est trop simple pour capturer la complexité
- Variance acceptable (généralise bien)

## 3. R-LEARNING

Points forts:

- ✓ Meilleur recall (~21% vs 3%)
- ✓ Fonction de récompense personnalisable
- ✓ Seuil de décision ajustable
- ✓ Adaptable aux besoins métier

Points faibles:

- X Temps d'entraînement long (~2-5 min)
- X Plus complexe à implémenter
- X Nécessite tuning des hyperparamètres
- X Accuracy globale plus faible (~87%)

Diagnostic Biais/Variance:

- Meilleur équilibre biais/variance
- Convergence de rho stable
- Moins d'underfitting sur classe minoritaire

## 4. RECOMMANDATIONS

Pour une mise en production RAPIDE:

- Utiliser Naive Bayes
- Avantage: Simplicité et rapidité

Pour OPTIMISER la détection de détresse:

- Utiliser R-Learning
- Avantage: Meilleur recall, moins de faux négatifs

APPROCHE HYBRIDE (Recommandée):

- Filtrage initial avec Naive Bayes

- Filtrage initial avec Naive Bayes
- Analyse approfondie avec R-Learning sur cas limites
- Combine rapidité et précision

## 5. AMÉLIORATIONS FUTURES

- ✓ Feature engineering (ratios, interactions)
- ✓ Tester Random Forest, XGBoost
- ✓ SMOTE pour oversampling plus sophistiqué
- ✓ Validation croisée k-fold
- ✓ Optimisation des seuils de décision
- ✓ Analyse SHAP pour interprétabilité

## 6. DIFFICULTÉS ET FAISABILITÉS

Difficultés rencontrées:

- Déséquilibre extrême des classes
- Tuning de la fonction de récompense R-Learning
- Trade-off precision/recall

Faisabilité:

- ✓ Les deux approches sont viables
- ✓ Naive Bayes: Production immédiate
- ✓ R-Learning: Nécessite expertise RL

FIN DE L'ANALYSE

""")

```
print("\n✓ Tous les graphiques ont été sauvegardés dans le répertoire courr
print("✓ Analyse complète terminée avec succès!")
print("\n" + "="*80)
```

## ÉTAPE 9: CONCLUSIONS ET RECOMMANDATIONS

### CONCLUSIONS FINALES

#### 1. ANALYSE DU PROBLÈME

- ✓ Dataset déséquilibré (93% sains vs 7% détresse)
- ✓ Problème de classification binaire
- ✓ Objectif: Minimiser les faux négatifs (coût élevé)

#### 2. NAIVE BAYES

Points forts:

- ✓ Très rapide à entraîner
- ✓ Bonne accuracy globale (~94%)
- ✓ Modèle simple et interprétable
- ✓ Faible variance (pas d'overfitting)

Points faibles:

Points faibles:

- X Très faible recall sur classe minoritaire (~3%)
- X Biais élevé sur la détection de détresse
- X UNDERFITTING sur la classe 1
- X Assume l'indépendance des features

Diagnostic Biais/Variance:

- BIAIS ÉLEVÉ (Underfitting)
- Le modèle est trop simple pour capturer la complexité
- Variance acceptable (généralise bien)

### 3. R-LEARNING

Points forts:

- ✓ Meilleur recall (~21% vs 3%)
- ✓ Fonction de récompense personnalisable
- ✓ Seuil de décision ajustable
- ✓ Adaptable aux besoins métier

Points faibles:

- X Temps d'entraînement long (~2-5 min)
- X Plus complexe à implémenter
- X Nécessite tuning des hyperparamètres
- X Accuracy globale plus faible (~87%)

Diagnostic Biais/Variance:

- Meilleur équilibre biais/variance
- Convergence de rho stable
- Moins d'underfitting sur classe minoritaire

### 4. RECOMMANDATIONS

Pour une mise en production RAPIDE:

- Utiliser Naive Bayes
- Avantage: Simplicité et rapidité

Commencez à coder ou à générer avec l'IA.

Commencez à coder ou à générer avec l'IA.

Commencez à coder ou à générer avec l'IA.

