Outliers: Détection et Traitement

Guide Complet pour Data Analyst Bancaire

---

1. Introduction

1.1 Qu'est-ce qu'un Outlier?

Un **outlier** (valeur aberrante) est une observation qui s'écarte significativement des autres observations dans un jeu de données.

Types d'outliers:

```
OUTLIERS
├── Outliers LÉGITIMES (vrais mais rares)
│   └── Ex: Client très fortuné, transaction exceptionnelle
│
├── Outliers ERREURS (données incorrectes)
│   └── Ex: Erreur de saisie (salaire 1,000,000 au lieu de 100,000)
│
└── Outliers FRAUDULEUX (anomalies suspectes)
    └── Ex: Transaction inhabituelle = possible fraude
```

1.2 Pourquoi c'est Critique en Banque?

Impact	Conséquence	Exemple
Statistiques biaisées	Moyenne faussée	Surestimer le solde moyen
Modèles dégradés	Prédictions erronées	Scoring de crédit incorrect
Fraude non détectée	Pertes financières	Transaction frauduleuse ratée
Faux positifs	Blocage clients légitimes	Carte bloquée injustement
Risque réglementaire	Non-conformité	Problèmes avec le régulateur

---

2. Méthodes de Détection

2.1 Vue d'Ensemble des Méthodes

```python
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

def detection_outliers_complete(df, colonne, methode='toutes'):
    """
    Détection d'outliers avec plusieurs méthodes
    """
    print("=" * 70)
```

```python
print(f"DÉTECTION D'OUTLIERS: {colonne}")
print("=" * 70)

serie = df[colonne].dropna()
n = len(serie)
resultats = {}

print(f"\nStatistiques de base:")
print(f"  N: {n}")
print(f"  Moyenne: {serie.mean():.2f}")
print(f"  Médiane: {serie.median():.2f}")
print(f"  Écart-type: {serie.std():.2f}")
print(f"  Min: {serie.min():.2f}")
print(f"  Max: {serie.max():.2f}")

# 1. MÉTHODE IQR (Interquartile Range)
print("\n" + "=" * 50)
print("1. MÉTHODE IQR (Tukey)")
print("=" * 50)

Q1 = serie.quantile(0.25)
Q3 = serie.quantile(0.75)
IQR = Q3 - Q1

lower_iqr = Q1 - 1.5 * IQR
upper_iqr = Q3 + 1.5 * IQR

outliers_iqr = serie[(serie < lower_iqr) | (serie > upper_iqr)]

print(f"  Q1: {Q1:.2f}")
print(f"  Q3: {Q3:.2f}")
print(f"  IQR: {IQR:.2f}")
print(f"  Borne inférieure: {lower_iqr:.2f}")
print(f"  Borne supérieure: {upper_iqr:.2f}")
print(f"  Outliers détectés: {len(outliers_iqr)} ({len(outliers_iqr)/n*100:.2f}%)")

resultats['IQR'] = {
    'bornes': (lower_iqr, upper_iqr),
    'n_outliers': len(outliers_iqr),
    'indices': outliers_iqr.index.tolist()
}

# 2. MÉTHODE Z-SCORE
print("\n" + "=" * 50)
print("2. MÉTHODE Z-SCORE")
print("=" * 50)

z_scores = np.abs(stats.zscore(serie))

for seuil in [2, 2.5, 3]:
    outliers_z = serie[z_scores > seuil]
    print(f"  Seuil |z| > {seuil}: {len(outliers_z)} outliers ({len(outliers_z)/n*100:

# Standard: seuil = 3
```

```python
outliers_zscore = serie[z_scores > 3]
resultats['Z-Score'] = {
    'seuil': 3,
    'n_outliers': len(outliers_zscore),
    'indices': outliers_zscore.index.tolist()
}

# 3. MÉTHODE MAD (Median Absolute Deviation)
print("\n" + "=" * 50)
print("3. MÉTHODE MAD (Robuste)")
print("=" * 50)

median = serie.median()
mad = np.median(np.abs(serie - median))

# Modified Z-score
modified_z = 0.6745 * (serie - median) / mad if mad != 0 else pd.Series(0, index=serie

outliers_mad = serie[np.abs(modified_z) > 3.5]

print(f"  Médiane: {median:.2f}")
print(f"  MAD: {mad:.2f}")
print(f"  Seuil |z_mod| > 3.5: {len(outliers_mad)} outliers ({len(outliers_mad)/n*100:

resultats['MAD'] = {
    'seuil': 3.5,
    'n_outliers': len(outliers_mad),
    'indices': outliers_mad.index.tolist()
}

# 4. MÉTHODE PERCENTILES
print("\n" + "=" * 50)
print("4. MÉTHODE PERCENTILES")
print("=" * 50)

for pct in [(1, 99), (2.5, 97.5), (5, 95)]:
    lower = serie.quantile(pct[0]/100)
    upper = serie.quantile(pct[1]/100)
    outliers_pct = serie[(serie < lower) | (serie > upper)]
    print(f"  Percentiles {pct}: {len(outliers_pct)} outliers")

# 5. VISUALISATION
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle(f'Détection d\'outliers: {colonne}', fontsize=14, fontweight='bold')

# Box plot
ax1 = axes[0, 0]
bp = ax1.boxplot(serie, vert=True, patch_artist=True)
bp['boxes'][0].set_facecolor('lightblue')
ax1.scatter([1]*len(outliers_iqr), outliers_iqr, color='red', s=50, zorder=5, label='O
ax1.set_title('Box Plot avec Outliers IQR')
ax1.legend()

# Histogramme avec bornes
```

```
    ax2 = axes[0, 1]
    serie.hist(ax=ax2, bins=50, edgecolor='black', alpha=0.7)
    ax2.axvline(lower_iqr, color='red', linestyle='--', label=f'Borne inf ({lower_iqr:.0f}
    ax2.axvline(upper_iqr, color='red', linestyle='--', label=f'Borne sup ({upper_iqr:.0f}
    ax2.legend()
    ax2.set_title('Histogramme avec bornes IQR')

    # Distribution des Z-scores
    ax3 = axes[1, 0]
    ax3.hist(z_scores, bins=50, edgecolor='black', alpha=0.7)
    ax3.axvline(3, color='red', linestyle='--', label='Seuil z=3')
    ax3.legend()
    ax3.set_title('Distribution des Z-scores')
    ax3.set_xlabel('|Z-score|')

    # Scatter plot avec indices
    ax4 = axes[1, 1]
    ax4.scatter(range(len(serie)), serie, alpha=0.3, s=10)
    ax4.scatter(
        [list(serie.index).index(i) for i in outliers_iqr.index if i in serie.index],
        outliers_iqr,
        color='red', s=50, label='Outliers IQR'
    )
    ax4.axhline(upper_iqr, color='red', linestyle='--', alpha=0.5)
    ax4.axhline(lower_iqr, color='red', linestyle='--', alpha=0.5)
    ax4.legend()
    ax4.set_title('Valeurs avec outliers marqués')
    ax4.set_xlabel('Index')
    ax4.set_ylabel(colonne)

    plt.tight_layout()
    plt.savefig(f'outliers_{colonne}.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 6. COMPARAISON DES MÉTHODES
    print("\n" + "=" * 50)
    print("COMPARAISON DES MÉTHODES")
    print("=" * 50)

    comparison = pd.DataFrame({
        'Méthode': ['IQR', 'Z-Score', 'MAD'],
        'N_Outliers': [resultats['IQR']['n_outliers'],
                       resultats['Z-Score']['n_outliers'],
                       resultats['MAD']['n_outliers']],
        'Pourcentage': [resultats['IQR']['n_outliers']/n*100,
                        resultats['Z-Score']['n_outliers']/n*100,
                        resultats['MAD']['n_outliers']/n*100]
    })
    print(comparison.to_string(index=False))

    return resultats

# Exemple bancaire
np.random.seed(42)
```

```
n = 1000

# Créer des données avec outliers
montants = np.concatenate([
    np.random.lognormal(9, 0.5, 950),  # Transactions normales
    np.random.lognormal(12, 0.3, 30),   # Transactions élevées légitimes
    np.array([1000000, 1500000, 2000000, 50, 75, 100,  # Outliers extrêmes
              -1000, -500, 0, 0, 0, 5000000, 8000000,  # Erreurs potentielles
              3000000, 4000000, 6000000, 7000000, 9000000, 10000000, 15000000])  # Plus d'
])
np.random.shuffle(montants)

df_transactions = pd.DataFrame({
    'montant': montants[:1000]
})

resultats = detection_outliers_complete(df_transactions, 'montant')
```

2.2 Méthodes Multivariées

```
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.covariance import EllipticEnvelope

def detection_multivariee(df, colonnes, contamination=0.05):
    """
    Détection d'outliers multivariés
    """
    print("=" * 70)
    print("DÉTECTION D'OUTLIERS MULTIVARIÉS")
    print("=" * 70)

    X = df[colonnes].dropna()
    n = len(X)

    print(f"\nVariables: {colonnes}")
    print(f"Observations: {n}")
    print(f"Taux de contamination attendu: {contamination*100}%")

    resultats = {}

    # 1. ISOLATION FOREST
    print("\n" + "=" * 50)
    print("1. ISOLATION FOREST")
    print("=" * 50)

    iso_forest = IsolationForest(
        contamination=contamination,
        random_state=42,
        n_estimators=100
    )

    labels_iso = iso_forest.fit_predict(X)
    scores_iso = iso_forest.decision_function(X)
```

```
outliers_iso = (labels_iso == -1).sum()
print(f"  Outliers détectés: {outliers_iso} ({outliers_iso/n*100:.2f}%)")

resultats['IsolationForest'] = {
    'labels': labels_iso,
    'scores': scores_iso,
    'n_outliers': outliers_iso
}

# 2. LOCAL OUTLIER FACTOR
print("\n" + "=" * 50)
print("2. LOCAL OUTLIER FACTOR (LOF)")
print("=" * 50)

lof = LocalOutlierFactor(
    n_neighbors=20,
    contamination=contamination
)

labels_lof = lof.fit_predict(X)
scores_lof = lof.negative_outlier_factor_

outliers_lof = (labels_lof == -1).sum()
print(f"  Outliers détectés: {outliers_lof} ({outliers_lof/n*100:.2f}%)")

resultats['LOF'] = {
    'labels': labels_lof,
    'scores': scores_lof,
    'n_outliers': outliers_lof
}

# 3. ELLIPTIC ENVELOPE (Mahalanobis)
print("\n" + "=" * 50)
print("3. ELLIPTIC ENVELOPE (Mahalanobis)")
print("=" * 50)

try:
    envelope = EllipticEnvelope(
        contamination=contamination,
        random_state=42
    )

    labels_env = envelope.fit_predict(X)
    scores_env = envelope.decision_function(X)

    outliers_env = (labels_env == -1).sum()
    print(f"  Outliers détectés: {outliers_env} ({outliers_env/n*100:.2f}%)")

    resultats['Elliptic'] = {
        'labels': labels_env,
        'scores': scores_env,
        'n_outliers': outliers_env
    }
```

```
except Exception as e:
    print(f"  Erreur: {e}")
    resultats['Elliptic'] = None

# 4. CONSENSUS (accord entre méthodes)
print("\n" + "=" * 50)
print("4. CONSENSUS")
print("=" * 50)

# Points identifiés comme outliers par au moins 2 méthodes
labels_all = np.vstack([labels_iso, labels_lof])
if resultats['Elliptic']:
    labels_all = np.vstack([labels_all, labels_env])

consensus = (labels_all == -1).sum(axis=0)

outliers_consensus_2 = (consensus >= 2).sum()
outliers_consensus_all = (consensus == len(labels_all)).sum()

print(f"  Outliers par au moins 2 méthodes: {outliers_consensus_2}")
print(f"  Outliers par TOUTES les méthodes: {outliers_consensus_all}")

resultats['consensus'] = consensus

# 5. VISUALISATION
if len(colonnes) >= 2:
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # Isolation Forest
    scatter1 = axes[0].scatter(X.iloc[:, 0], X.iloc[:, 1], c=labels_iso,
                               cmap='RdYlGn', alpha=0.5)
    axes[0].set_title('Isolation Forest')
    axes[0].set_xlabel(colonnes[0])
    axes[0].set_ylabel(colonnes[1])

    # LOF
    scatter2 = axes[1].scatter(X.iloc[:, 0], X.iloc[:, 1], c=labels_lof,
                               cmap='RdYlGn', alpha=0.5)
    axes[1].set_title('Local Outlier Factor')
    axes[1].set_xlabel(colonnes[0])
    axes[1].set_ylabel(colonnes[1])

    # Consensus
    scatter3 = axes[2].scatter(X.iloc[:, 0], X.iloc[:, 1], c=consensus,
                               cmap='Reds', alpha=0.5)
    axes[2].set_title('Consensus (rouge = plus de votes)')
    axes[2].set_xlabel(colonnes[0])
    axes[2].set_ylabel(colonnes[1])
    plt.colorbar(scatter3, ax=axes[2], label='Nb méthodes')

    plt.tight_layout()
    plt.savefig('outliers_multivaries.png', dpi=300)
    plt.show()
```

```
    return resultats

# Exemple: Détection de comportements anormaux
np.random.seed(42)
n = 1000

df_comportement = pd.DataFrame({
    'montant_moyen': np.concatenate([
        np.random.lognormal(8, 0.5, 950),
        np.random.lognormal(12, 0.3, 50)
    ]),
    'nb_transactions': np.concatenate([
        np.random.poisson(15, 950),
        np.random.poisson(100, 50)
    ]),
    'nb_pays': np.concatenate([
        np.random.poisson(2, 950),
        np.random.poisson(10, 50)
    ])
})

resultats_multi = detection_multivariee(
    df_comportement,
    ['montant_moyen', 'nb_transactions', 'nb_pays'],
    contamination=0.05
)
```

2.3 Détection Contextuelle (Bancaire)

```
def detection_outliers_contextuelle(df, col_valeur, col_groupe):
    """
    Détection d'outliers par groupe (contexte)

    Exemple: Un montant élevé pour un client "basique"
    n'est pas élevé pour un client "premium"
    """
    print("=" * 70)
    print(f"DÉTECTION CONTEXTUELLE: {col_valeur} par {col_groupe}")
    print("=" * 70)

    resultats = []

    for groupe, data in df.groupby(col_groupe):
        serie = data[col_valeur]

        # IQR par groupe
        Q1 = serie.quantile(0.25)
        Q3 = serie.quantile(0.75)
        IQR = Q3 - Q1

        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR

        outliers = serie[(serie < lower) | (serie > upper)]
```

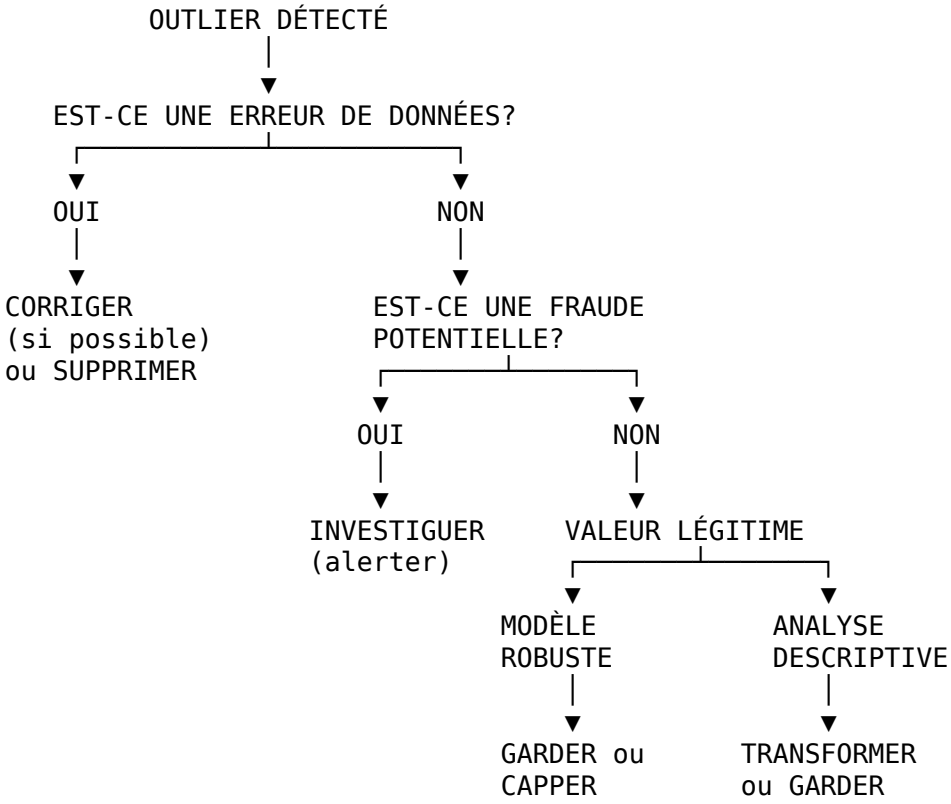
```
            resultats.append({
                'Groupe': groupe,
                'N': len(serie),
                'Moyenne': serie.mean(),
                'Borne_inf': lower,
                'Borne_sup': upper,
                'N_outliers': len(outliers),
                'Pct_outliers': len(outliers)/len(serie)*100
            })

            # Marquer les outliers dans le DataFrame original
            df.loc[outliers.index, 'is_outlier_contextuel'] = True

    df['is_outlier_contextuel'] = df['is_outlier_contextuel'].fillna(False)

    print("\nRésultats par groupe:")
    df_resultats = pd.DataFrame(resultats)
    print(df_resultats.to_string(index=False))

    # Visualisation
    fig, ax = plt.subplots(figsize=(12, 6))

    positions = []
    for i, groupe in enumerate(df[col_groupe].unique()):
        data = df[df[col_groupe] == groupe][col_valeur]
        outliers = df[(df[col_groupe] == groupe) & (df['is_outlier_contextuel'])][col_vale

        bp = ax.boxplot(data, positions=[i], widths=0.6, patch_artist=True)
        bp['boxes'][0].set_facecolor(f'C{i}')

        if len(outliers) > 0:
            ax.scatter([i]*len(outliers), outliers, color='red', s=50, zorder=5)

        positions.append(i)

    ax.set_xticks(positions)
    ax.set_xticklabels(df[col_groupe].unique())
    ax.set_ylabel(col_valeur)
    ax.set_title(f'Outliers contextuels: {col_valeur} par {col_groupe}')

    plt.tight_layout()
    plt.savefig('outliers_contextuels.png', dpi=300)
    plt.show()

    return df, df_resultats

# Exemple: Transactions par segment client
np.random.seed(42)

df_segments = pd.DataFrame({
    'segment': np.random.choice(['Premium', 'Standard', 'Basique'], 1000, p=[0.2, 0.5, 0.3
    'montant_transaction': np.zeros(1000)
})
```

```python
for seg in ['Premium', 'Standard', 'Basique']:
    mask = df_segments['segment'] == seg
    n_seg = mask.sum()

    if seg == 'Premium':
        df_segments.loc[mask, 'montant_transaction'] = np.random.lognormal(10, 0.5, n_seg)
    elif seg == 'Standard':
        df_segments.loc[mask, 'montant_transaction'] = np.random.lognormal(8, 0.6, n_seg)
    else:
        df_segments.loc[mask, 'montant_transaction'] = np.random.lognormal(7, 0.7, n_seg)

df_segments, resultats_ctx = detection_outliers_contextuelle(
    df_segments, 'montant_transaction', 'segment'
)
```

---

3. Traitement des Outliers

3.1 Arbre de Décision pour le Traitement



3.2 Méthodes de Traitement

```python
def traiter_outliers(df, colonne, methode='winsorize', params=None):
    """
    Différentes méthodes de traitement des outliers
```

```
Méthodes:
- suppression: Supprimer les outliers
- winsorize: Ramener aux bornes (capping)
- log_transform: Transformation logarithmique
- imputation: Remplacer par médiane
- flag: Juste marquer (ne pas modifier)
"""
print("=" * 70)
print(f"TRAITEMENT DES OUTLIERS: {colonne}")
print(f"Méthode: {methode}")
print("=" * 70)

df_out = df.copy()
serie = df_out[colonne]

# Calculer les bornes IQR
Q1 = serie.quantile(0.25)
Q3 = serie.quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

mask_outliers = (serie < lower) | (serie > upper)
n_outliers = mask_outliers.sum()

print(f"\nOutliers détectés: {n_outliers} ({n_outliers/len(serie)*100:.2f}%)")
print(f"Bornes: [{lower:.2f}, {upper:.2f}]")

# Stats avant
print(f"\nAVANT traitement:")
print(f"  Moyenne: {serie.mean():.2f}")
print(f"  Médiane: {serie.median():.2f}")
print(f"  Min: {serie.min():.2f}")
print(f"  Max: {serie.max():.2f}")

if methode == 'suppression':
    # Supprimer les lignes avec outliers
    df_out = df_out[~mask_outliers]
    print(f"\nLignes supprimées: {n_outliers}")

elif methode == 'winsorize':
    # Ramener aux bornes
    percentile_lower = params.get('lower', 5) if params else 5
    percentile_upper = params.get('upper', 95) if params else 95

    lower_val = serie.quantile(percentile_lower/100)
    upper_val = serie.quantile(percentile_upper/100)

    df_out[colonne] = serie.clip(lower=lower_val, upper=upper_val)
    print(f"\nValeurs cappées à [{lower_val:.2f}, {upper_val:.2f}]")
    print(f"  (Percentiles {percentile_lower}-{percentile_upper})")

elif methode == 'log_transform':
    # Transformation logarithmique
```

```
        min_val = serie[serie > 0].min()
        df_out[f'{colonne}_log'] = np.log1p(serie.clip(lower=0))
        print(f"\nNouvelle colonne créée: {colonne}_log")

    elif methode == 'imputation':
        # Remplacer par la médiane
        median_val = serie[~mask_outliers].median()
        df_out.loc[mask_outliers, colonne] = median_val
        print(f"\nOutliers remplacés par la médiane: {median_val:.2f}")

    elif methode == 'flag':
        # Créer une colonne indicateur
        df_out[f'{colonne}_outlier'] = mask_outliers.astype(int)
        print(f"\nColonne indicateur créée: {colonne}_outlier")

    else:
        raise ValueError(f"Méthode inconnue: {methode}")

    # Stats après
    if methode != 'flag' and methode != 'log_transform':
        print(f"\nAPRÈS traitement:")
        print(f"  Moyenne: {df_out[colonne].mean():.2f}")
        print(f"  Médiane: {df_out[colonne].median():.2f}")
        print(f"  Min: {df_out[colonne].min():.2f}")
        print(f"  Max: {df_out[colonne].max():.2f}")

    return df_out

# Démonstration des différentes méthodes
np.random.seed(42)
df_demo = pd.DataFrame({
    'revenu': np.concatenate([
        np.random.lognormal(10, 0.5, 980),
        np.array([1000000, 2000000, 3000000, 5000000, 10000000,  # Très hauts revenus
                  100, 200, 50, 75, 150, 80, 120, 90, 60, 40])   # Très bas revenus
    ])
})

# Tester chaque méthode
print("\n" + "="*80)
print("COMPARAISON DES MÉTHODES")
print("="*80)

for methode in ['suppression', 'winsorize', 'log_transform', 'imputation', 'flag']:
    df_traite = traiter_outliers(df_demo.copy(), 'revenu', methode=methode)
    print("\n" + "-"*50)
```

3.3 Impact sur les Modèles

```
def comparer_impact_outliers(df, col_features, col_target):
    """
    Comparer l'impact du traitement des outliers sur les modèles
    """
    from sklearn.model_selection import cross_val_score
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

print("=" * 70)
print("IMPACT DES OUTLIERS SUR LES MODÈLES")
print("=" * 70)

resultats = []

# 1. Données originales
X_orig = df[col_features]
y_orig = df[col_target]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_orig)

for model_name, model in [('Logistic', LogisticRegression(max_iter=1000)),
                          ('RandomForest', RandomForestClassifier(n_estimators=50, ra
    scores = cross_val_score(model, X_scaled, y_orig, cv=5, scoring='roc_auc')
    resultats.append({
        'Données': 'Originales',
        'Modèle': model_name,
        'AUC_mean': scores.mean(),
        'AUC_std': scores.std()
    })

# 2. Données sans outliers
df_no_out = df.copy()
for col in col_features:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df_no_out = df_no_out[(df_no_out[col] >= lower) & (df_no_out[col] <= upper)]

X_no_out = df_no_out[col_features]
y_no_out = df_no_out[col_target]
X_no_out_scaled = StandardScaler().fit_transform(X_no_out)

for model_name, model in [('Logistic', LogisticRegression(max_iter=1000)),
                          ('RandomForest', RandomForestClassifier(n_estimators=50, ra
    scores = cross_val_score(model, X_no_out_scaled, y_no_out, cv=5, scoring='roc_auc'
    resultats.append({
        'Données': 'Sans outliers',
        'Modèle': model_name,
        'AUC_mean': scores.mean(),
        'AUC_std': scores.std()
    })

# 3. Données winsorisées
df_wins = df.copy()
for col in col_features:
```

```
        lower = df[col].quantile(0.05)
        upper = df[col].quantile(0.95)
        df_wins[col] = df_wins[col].clip(lower=lower, upper=upper)

    X_wins = df_wins[col_features]
    y_wins = df_wins[col_target]
    X_wins_scaled = StandardScaler().fit_transform(X_wins)

    for model_name, model in [('Logistic', LogisticRegression(max_iter=1000)),
                              ('RandomForest', RandomForestClassifier(n_estimators=50, ra
        scores = cross_val_score(model, X_wins_scaled, y_wins, cv=5, scoring='roc_auc')
        resultats.append({
            'Données': 'Winsorisées',
            'Modèle': model_name,
            'AUC_mean': scores.mean(),
            'AUC_std': scores.std()
        })

    # Résumé
    df_resultats = pd.DataFrame(resultats)

    print("\nComparaison:")
    pivot = df_resultats.pivot(index='Données', columns='Modèle', values='AUC_mean')
    print(pivot.round(4))

    # Visualisation
    fig, ax = plt.subplots(figsize=(10, 6))

    x = np.arange(len(pivot.index))
    width = 0.35

    ax.bar(x - width/2, pivot['Logistic'], width, label='Logistic')
    ax.bar(x + width/2, pivot['RandomForest'], width, label='RandomForest')

    ax.set_ylabel('AUC-ROC')
    ax.set_xlabel('Traitement des données')
    ax.set_title('Impact du traitement des outliers sur la performance')
    ax.set_xticks(x)
    ax.set_xticklabels(pivot.index)
    ax.legend()
    ax.set_ylim(0.5, 1.0)

    plt.tight_layout()
    plt.savefig('impact_outliers_modeles.png', dpi=300)
    plt.show()

    return df_resultats

# Exemple
np.random.seed(42)
n = 1000

df_model = pd.DataFrame({
    'revenu': np.concatenate([np.random.lognormal(10, 0.5, 980),
```

```
                                np.random.lognormal(13, 0.3, 20)]),
    'score_bureau': np.concatenate([np.random.randint(400, 800, 980),
                                    np.random.randint(200, 400, 20)]),
    'anciennete': np.concatenate([np.random.exponential(5, 980),
                                  np.random.exponential(0.5, 20)]),
    'defaut': np.random.choice([0, 1], 1000, p=[0.93, 0.07])
})

impact = comparer_impact_outliers(df_model, ['revenu', 'score_bureau', 'anciennete'], 'def
```

---

4. Cas Pratiques Bancaires

4.1 Détection de Fraude

```
def detection_fraude_outliers(df_transactions):
    """
    Système de détection de fraude basé sur les outliers
    """
    print("=" * 70)
    print("SYSTÈME DE DÉTECTION DE FRAUDE")
    print("=" * 70)

    df = df_transactions.copy()

    # 1. Créer des features comportementales
    print("\n1. CRÉATION DES FEATURES")
    print("-" * 50)

    # Ratio montant vs moyenne client
    df['ratio_montant_moyen'] = df.groupby('client_id')['montant'].transform(
        lambda x: x / x.mean()
    )

    # Distance temporelle depuis dernière transaction
    df = df.sort_values(['client_id', 'date'])
    df['heures_depuis_derniere'] = df.groupby('client_id')['date'].diff().dt.total_seconds

    # Nombre de transactions dans les dernières 24h
    # (simplifié pour l'exemple)

    print("  Features créées: ratio_montant_moyen, heures_depuis_derniere")

    # 2. Détection par plusieurs méthodes
    print("\n2. DÉTECTION MULTI-MÉTHODES")
    print("-" * 50)

    df['score_fraude'] = 0

    # Méthode 1: Montant anormalement élevé (par client)
    for client_id, group in df.groupby('client_id'):
        Q1 = group['montant'].quantile(0.25)
        Q3 = group['montant'].quantile(0.75)
```

```
        IQR = Q3 - Q1
        upper = Q3 + 2 * IQR  # Seuil plus strict

        mask = (df['client_id'] == client_id) & (df['montant'] > upper)
        df.loc[mask, 'score_fraude'] += 1

    # Méthode 2: Transaction trop rapide après la précédente
    mask_rapid = df['heures_depuis_derniere'] < 0.5  # Moins de 30 minutes
    df.loc[mask_rapid, 'score_fraude'] += 1

    # Méthode 3: Ratio montant élevé
    mask_ratio = df['ratio_montant_moyen'] > 5
    df.loc[mask_ratio, 'score_fraude'] += 1

    # Méthode 4: Isolation Forest global
    features_fraude = ['montant', 'ratio_montant_moyen']
    X_fraude = df[features_fraude].dropna()

    iso = IsolationForest(contamination=0.02, random_state=42)
    labels_iso = iso.fit_predict(X_fraude)
    df.loc[X_fraude.index, 'score_fraude'] += (labels_iso == -1).astype(int)

    # 3. Classification finale
    print("\n3. RÉSULTATS")
    print("-" * 50)

    df['alerte_fraude'] = df['score_fraude'] >= 2

    n_alertes = df['alerte_fraude'].sum()
    print(f"  Alertes générées: {n_alertes} ({n_alertes/len(df)*100:.2f}%)")

    # Distribution des scores
    print("\n  Distribution des scores de fraude:")
    print(df['score_fraude'].value_counts().sort_index())

    # Top transactions suspectes
    print("\n  Top 10 transactions les plus suspectes:")
    top_suspectes = df.nlargest(10, 'score_fraude')[['client_id', 'montant', 'score_fraude
    print(top_suspectes.to_string(index=False))

    return df

# Simuler des transactions
np.random.seed(42)
n_clients = 100
n_transactions = 5000

df_tx = pd.DataFrame({
    'tx_id': range(n_transactions),
    'client_id': np.random.randint(0, n_clients, n_transactions),
    'date': pd.date_range('2024-01-01', periods=n_transactions, freq='15T'),
    'montant': np.random.lognormal(7, 1, n_transactions)
})
```

```python
# Ajouter des transactions frauduleuses
n_fraudes = 50
idx_fraudes = np.random.choice(n_transactions, n_fraudes, replace=False)
df_tx.loc[idx_fraudes, 'montant'] = df_tx.loc[idx_fraudes, 'montant'] * np.random.uniform(

df_avec_alertes = detection_fraude_outliers(df_tx)
```

4.2 Nettoyage pour Scoring

```python
def nettoyer_pour_scoring(df, colonnes_numeriques, target):
    """
    Pipeline de nettoyage des outliers pour un modèle de scoring
    """
    print("=" * 70)
    print("NETTOYAGE DES DONNÉES POUR SCORING")
    print("=" * 70)

    df_clean = df.copy()
    rapport = []

    for col in colonnes_numeriques:
        print(f"\n--- {col} ---")

        # Stats initiales
        n_initial = df_clean[col].notna().sum()

        # Détecter outliers
        Q1 = df_clean[col].quantile(0.25)
        Q3 = df_clean[col].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - 3 * IQR  # Seuil plus large (3 IQR)
        upper = Q3 + 3 * IQR

        mask_outliers = (df_clean[col] < lower) | (df_clean[col] > upper)
        n_outliers = mask_outliers.sum()

        print(f"  Outliers détectés: {n_outliers}")

        # Décision de traitement
        if n_outliers / n_initial > 0.1:
            # Plus de 10% d'outliers: winsorize
            traitement = 'winsorize'
            p5 = df_clean[col].quantile(0.05)
            p95 = df_clean[col].quantile(0.95)
            df_clean[col] = df_clean[col].clip(lower=p5, upper=p95)
            print(f"  Traitement: Winsorize à [P5, P95]")
        elif n_outliers > 0:
            # Quelques outliers: garder et flagger
            traitement = 'flag'
            df_clean[f'{col}_outlier'] = mask_outliers.astype(int)
            print(f"  Traitement: Flag créé")
        else:
            traitement = 'aucun'
            print(f"  Traitement: Aucun")
```

```
            rapport.append({
                'Variable': col,
                'N_outliers': n_outliers,
                'Pct_outliers': n_outliers/n_initial*100,
                'Traitement': traitement
            })

    # Résumé
    print("\n" + "=" * 70)
    print("RAPPORT DE NETTOYAGE")
    print("=" * 70)

    df_rapport = pd.DataFrame(rapport)
    print(df_rapport.to_string(index=False))

    return df_clean, df_rapport

# Application
np.random.seed(42)
n = 2000

df_scoring = pd.DataFrame({
    'revenu': np.concatenate([np.random.lognormal(10, 0.5, 1900),
                              np.random.lognormal(14, 0.2, 100)]),
    'age': np.concatenate([np.random.randint(25, 65, 1950),
                           np.array([15, 16, 17, 18, 95, 96, 97, 98] * 6 + [12, 13])]),
    'anciennete': np.concatenate([np.random.exponential(5, 1980),
                                  np.array([50, 60, 70, 80] * 5)]),
    'nb_credits': np.concatenate([np.random.poisson(2, 1980),
                                  np.array([15, 20, 25, 30] * 5)]),
    'defaut': np.random.choice([0, 1], 2000, p=[0.95, 0.05])
})

df_scoring_clean, rapport = nettoyer_pour_scoring(
    df_scoring,
    ['revenu', 'age', 'anciennete', 'nb_credits'],
    'defaut'
)
```

---

5. Résumé et Bonnes Pratiques

5.1 Tableau Récapitulatif des Méthodes

Méthode	Type	Robuste?	Quand utiliser
IQR	Univarié	Oui	Données non normales
Z-Score	Univarié	Non	Données ~ normales
MAD	Univarié	Oui	Petits échantillons
Percentiles	Univarié	Oui	Winsorization
Isolation Forest	Multivarié	Oui	Fraude, anomalies
LOF	Multivarié	Oui	Données denses

Méthode	Type	Robuste?	Quand utiliser
Mahalanobis	Multivarié	Non	Données normales

5.2 Checklist

```
AVANT DE TRAITER:
□ Documenter le nombre et % d'outliers
□ Analyser visuellement (box plots, scatter)
□ Identifier le type (erreur, fraude, légitime)
□ Vérifier le contexte métier

PENDANT LE TRAITEMENT:
□ Choisir la méthode adaptée au contexte
□ Ne jamais supprimer aveuglément
□ Conserver les outliers pour analyse séparée
□ Créer des indicateurs si pertinent

APRÈS LE TRAITEMENT:
□ Comparer les statistiques avant/après
□ Vérifier l'impact sur le modèle
□ Documenter les choix effectués
□ Surveiller en production
```

5.3 Mnémotechniques

“IQR = 1.5 Interquartile” - Borne inf = Q1 - 1.5 × IQR - Borne sup = Q3 + 1.5 × IQR

“WISE” pour le traitement: - **W**insorize (capper) - **I**mputer (remplacer) - **S**upprimer (si erreur) - **E**xclure (pour analyse séparée)

“3 Sigma” pour Z-score: - $|z| > 3$ = outlier (0.3% des données si normal)

5.4 Règles d’Or en Banque

1. **Ne jamais supprimer automatiquement** - Les outliers peuvent être des fraudes
2. **Contextualiser** - Un outlier pour un segment peut être normal pour un autre
3. **Documenter** - Le régulateur peut demander des explications
4. **Valider métier** - Impliquer les experts avant de traiter
5. **Surveiller** - Les nouveaux outliers peuvent signaler des problèmes