Prévisions: Analyse Univariée et Multivariée

Guide Complet pour Data Analyst Bancaire

---

1. Introduction

1.1 Définitions

Analyse Univariée: Étude d'UNE seule variable à la fois - Objectif: Comprendre la distribution, détecter les anomalies - Outils: Statistiques descriptives, histogrammes, box plots

Analyse Bivariée: Étude de la RELATION entre DEUX variables - Objectif: Identifier les corrélations, dépendances - Outils: Scatter plots, corrélation, tests statistiques

Analyse Multivariée: Étude SIMULTANÉE de PLUSIEURS variables - Objectif: Modéliser des phénomènes complexes, prédire - Outils: Régression multiple, ACP, clustering

1.2 Contexte Bancaire

Type d'analyse	Application bancaire	Exemple
Univariée	Monitoring KPIs	Distribution des soldes
Bivariée	Facteurs de risque	Revenu vs Défaut
Multivariée	Scoring crédit	Modèle de PD

---

2. Analyse Univariée en Profondeur

2.1 Statistiques Descriptives Complètes

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

def analyse_univariee_complete(serie, nom_variable="Variable"):
    """
    Analyse univariée exhaustive d'une variable
    """
    print("=" * 70)
    print(f"ANALYSE UNIVARIÉE: {nom_variable}")
    print("=" * 70)

    # 1. Statistiques de base
    print("\n1. STATISTIQUES DE BASE")
    print("-" * 40)

    stats_dict = {
        'N (total)': len(serie),
        'N (valides)': serie.count(),
```

```
    'N (manquants)': serie.isnull().sum(),
    '% manquants': f"{serie.isnull().mean()*100:.2f}%"
}

if serie.dtype in ['int64', 'float64']:
    stats_dict.update({
        'Minimum': serie.min(),
        'Maximum': serie.max(),
        'Étendue': serie.max() - serie.min(),
        'Moyenne': serie.mean(),
        'Médiane': serie.median(),
        'Mode': serie.mode().iloc[0] if len(serie.mode()) > 0 else np.nan,
        'Écart-type': serie.std(),
        'Variance': serie.var(),
        'CV (%)': f"{(serie.std()/serie.mean())*100:.2f}%" if serie.mean() != 0 else '
        'Q1 (25%)': serie.quantile(0.25),
        'Q3 (75%)': serie.quantile(0.75),
        'IQR': serie.quantile(0.75) - serie.quantile(0.25),
        'Skewness': serie.skew(),
        'Kurtosis': serie.kurtosis()
    })

for key, value in stats_dict.items():
    print(f"  {key}: {value}")

# 2. Interprétation de la forme
print("\n2. INTERPRÉTATION DE LA FORME")
print("-" * 40)

if serie.dtype in ['int64', 'float64']:
    skew = serie.skew()
    kurt = serie.kurtosis()

    # Asymétrie
    if abs(skew) < 0.5:
        asymetrie = "Symétrique"
    elif skew > 0:
        asymetrie = "Asymétrie positive (queue à droite)"
    else:
        asymetrie = "Asymétrie négative (queue à gauche)"

    # Aplatissement
    if abs(kurt) < 0.5:
        aplatissement = "Mésokurtique (normal)"
    elif kurt > 0:
        aplatissement = "Leptokurtique (queues épaisses)"
    else:
        aplatissement = "Platykurtique (queues fines)"

    print(f"  Asymétrie (skew={skew:.3f}): {asymetrie}")
    print(f"  Aplatissement (kurt={kurt:.3f}): {aplatissement}")

    # Test de normalité
    if len(serie.dropna()) >= 8:
```

```
            stat, p_value = stats.shapiro(serie.dropna()[:5000])  # Limite à 5000
            print(f"\n  Test de Shapiro-Wilk:")
            print(f"    Statistique: {stat:.4f}")
            print(f"    p-value: {p_value:.4f}")
            if p_value < 0.05:
                print("    => Distribution NON normale")
            else:
                print("    => Distribution approximativement normale")

    # 3. Détection des outliers
    print("\n3. DÉTECTION DES OUTLIERS")
    print("-" * 40)

    if serie.dtype in ['int64', 'float64']:
        Q1 = serie.quantile(0.25)
        Q3 = serie.quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        outliers = serie[(serie < lower_bound) | (serie > upper_bound)]

        print(f"  Méthode IQR:")
        print(f"    Borne inférieure: {lower_bound:.2f}")
        print(f"    Borne supérieure: {upper_bound:.2f}")
        print(f"    Nombre d'outliers: {len(outliers)} ({len(outliers)/len(serie)*100:.2f}

        # Méthode Z-score
        z_scores = np.abs(stats.zscore(serie.dropna()))
        outliers_zscore = (z_scores > 3).sum()
        print(f"\n  Méthode Z-score (|z| > 3):")
        print(f"    Nombre d'outliers: {outliers_zscore}")

    return stats_dict

# Application bancaire
np.random.seed(42)
soldes_comptes = np.concatenate([
    np.random.lognormal(10, 1, 4500),  # Distribution normale
    np.random.lognormal(12, 0.5, 500)   # Clients fortunés
])

resultats = analyse_univariee_complete(
    pd.Series(soldes_comptes),
    "Solde des Comptes (HTG)"
)
```

2.2 Visualisation Univariée

```
def visualiser_univariee(serie, nom_variable="Variable"):
    """
    Visualisation complète d'une variable numérique
    """
```

```python
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle(f'Analyse Univariée: {nom_variable}', fontsize=14, fontweight='bold')

# 1. Histogramme avec KDE
ax1 = axes[0, 0]
serie.hist(ax=ax1, bins=30, density=True, alpha=0.7, edgecolor='black')
serie.plot.kde(ax=ax1, color='red', linewidth=2)
ax1.axvline(serie.mean(), color='green', linestyle='--', label=f'Moyenne: {serie.mean(
ax1.axvline(serie.median(), color='orange', linestyle='--', label=f'Médiane: {serie.me
ax1.legend()
ax1.set_title('Distribution')
ax1.set_xlabel(nom_variable)

# 2. Box Plot
ax2 = axes[0, 1]
bp = ax2.boxplot(serie.dropna(), vert=True, patch_artist=True)
bp['boxes'][0].set_facecolor('lightblue')
ax2.set_title('Box Plot')
ax2.set_ylabel(nom_variable)

# 3. Violin Plot
ax3 = axes[0, 2]
parts = ax3.violinplot(serie.dropna(), showmeans=True, showmedians=True)
ax3.set_title('Violin Plot')
ax3.set_ylabel(nom_variable)

# 4. QQ Plot
ax4 = axes[1, 0]
stats.probplot(serie.dropna(), dist="norm", plot=ax4)
ax4.set_title('QQ Plot (Normalité)')

# 5. CDF (Fonction de répartition)
ax5 = axes[1, 1]
sorted_data = np.sort(serie.dropna())
cdf = np.arange(1, len(sorted_data) + 1) / len(sorted_data)
ax5.plot(sorted_data, cdf, linewidth=2)
ax5.axhline(0.5, color='red', linestyle='--', alpha=0.5)
ax5.axvline(serie.median(), color='red', linestyle='--', alpha=0.5)
ax5.set_title('Fonction de Répartition (CDF)')
ax5.set_xlabel(nom_variable)
ax5.set_ylabel('Probabilité cumulative')

# 6. Statistiques textuelles
ax6 = axes[1, 2]
ax6.axis('off')
stats_text = f"""
STATISTIQUES CLÉS

N = {len(serie)}
Moyenne = {serie.mean():.2f}
Médiane = {serie.median():.2f}
Écart-type = {serie.std():.2f}

Min = {serie.min():.2f}
```

```
    Max = {serie.max():.2f}

    Q1 = {serie.quantile(0.25):.2f}
    Q3 = {serie.quantile(0.75):.2f}

    Skewness = {serie.skew():.3f}
    Kurtosis = {serie.kurtosis():.3f}
    """
    ax6.text(0.1, 0.5, stats_text, fontsize=11, family='monospace',
             verticalalignment='center', bbox=dict(boxstyle='round', facecolor='wheat'))

    plt.tight_layout()
    plt.savefig(f'univariee_{nom_variable.replace(" ", "_")}.png', dpi=300, bbox_inches='t
    plt.show()

visualiser_univariee(pd.Series(soldes_comptes), "Solde des Comptes")
```

2.3 Prévision Univariée: Séries Temporelles

```
def prevision_univariee_bancaire(df, col_date, col_valeur, horizon=12):
    """
    Prévision univariée pour données bancaires (dépôts, prêts, etc.)

    Méthodes:
    1. Moyenne mobile
    2. Lissage exponentiel
    3. Décomposition saisonnière
    """
    from statsmodels.tsa.seasonal import seasonal_decompose
    from statsmodels.tsa.holtwinters import ExponentialSmoothing

    # Préparer les données
    df_ts = df[[col_date, col_valeur]].copy()
    df_ts[col_date] = pd.to_datetime(df_ts[col_date])
    df_ts.set_index(col_date, inplace=True)
    df_ts = df_ts.resample('M').sum()  # Agrégation mensuelle

    serie = df_ts[col_valeur]

    print("=" * 60)
    print(f"PRÉVISION UNIVARIÉE: {col_valeur}")
    print("=" * 60)

    # 1. Décomposition
    print("\n1. DÉCOMPOSITION DE LA SÉRIE")
    print("-" * 40)

    if len(serie) >= 24:
        decomposition = seasonal_decompose(serie, model='additive', period=12)

        fig, axes = plt.subplots(4, 1, figsize=(12, 10))
        decomposition.observed.plot(ax=axes[0], title='Série originale')
        decomposition.trend.plot(ax=axes[1], title='Tendance')
        decomposition.seasonal.plot(ax=axes[2], title='Saisonnalité')
```

```
    decomposition.resid.plot(ax=axes[3], title='Résidus')
    plt.tight_layout()
    plt.savefig(f'decomposition_{col_valeur}.png', dpi=300)
    plt.show()

# 2. Lissage exponentiel (Holt-Winters)
print("\n2. PRÉVISION HOLT-WINTERS")
print("-" * 40)

try:
    model = ExponentialSmoothing(
        serie,
        seasonal_periods=12,
        trend='add',
        seasonal='add'
    )
    fitted = model.fit()

    # Prévisions
    forecast = fitted.forecast(horizon)

    # Visualisation
    fig, ax = plt.subplots(figsize=(12, 6))
    serie.plot(ax=ax, label='Historique', linewidth=2)
    forecast.plot(ax=ax, label='Prévision', linestyle='--', linewidth=2, color='red')
    ax.fill_between(
        forecast.index,
        forecast * 0.9,
        forecast * 1.1,
        alpha=0.2,
        color='red',
        label='IC 90%'
    )
    ax.legend()
    ax.set_title(f'Prévision {col_valeur} - Holt-Winters')
    ax.set_xlabel('Date')
    ax.set_ylabel(col_valeur)
    plt.tight_layout()
    plt.savefig(f'prevision_{col_valeur}.png', dpi=300)
    plt.show()

    print(f"\nPrévisions ({horizon} mois):")
    print(forecast.to_frame(name='Prévision'))

    # Métriques
    print(f"\nMétriques du modèle:")
    print(f"  AIC: {fitted.aic:.2f}")
    print(f"  BIC: {fitted.bic:.2f}")

    return forecast

except Exception as e:
    print(f"Erreur: {e}")
    return None
```

```python
# Exemple: Prévision des dépôts mensuels
dates = pd.date_range('2020-01-01', periods=48, freq='M')
depots = 1000000 + 50000 * np.arange(48) + 100000 * np.sin(np.arange(48) * np.pi / 6) + np

df_depots = pd.DataFrame({
    'date': dates,
    'depots_mensuels': depots
})

previsions = prevision_univariee_bancaire(df_depots, 'date', 'depots_mensuels', horizon=12
```

---

3. Analyse Bivariée

3.1 Corrélation et Tests

```python
def analyse_bivariee(df, var1, var2):
    """
    Analyse bivariée complète entre deux variables
    """
    print("=" * 70)
    print(f"ANALYSE BIVARIÉE: {var1} vs {var2}")
    print("=" * 70)

    x = df[var1].dropna()
    y = df[var2].dropna()

    # Aligner les données
    common_idx = x.index.intersection(y.index)
    x = x.loc[common_idx]
    y = y.loc[common_idx]

    # 1. Types de variables
    x_numeric = x.dtype in ['int64', 'float64']
    y_numeric = y.dtype in ['int64', 'float64']

    print("\n1. TYPES DE VARIABLES")
    print("-" * 40)
    print(f"  {var1}: {'Numérique' if x_numeric else 'Catégorielle'}")
    print(f"  {var2}: {'Numérique' if y_numeric else 'Catégorielle'}")

    # 2. Analyse selon les types
    print("\n2. MESURES D'ASSOCIATION")
    print("-" * 40)

    if x_numeric and y_numeric:
        # Deux numériques: Corrélation
        pearson_r, pearson_p = stats.pearsonr(x, y)
        spearman_r, spearman_p = stats.spearmanr(x, y)

        print(f"  Corrélation de Pearson: r = {pearson_r:.4f} (p = {pearson_p:.4f})")
        print(f"  Corrélation de Spearman: rho = {spearman_r:.4f} (p = {spearman_p:.4f})")
```

```
        # Interprétation
        if abs(pearson_r) < 0.3:
            force = "Faible"
        elif abs(pearson_r) < 0.7:
            force = "Modérée"
        else:
            force = "Forte"

        direction = "positive" if pearson_r > 0 else "négative"
        print(f"\n  Interprétation: Relation {force} et {direction}")

        # Visualisation
        fig, axes = plt.subplots(1, 3, figsize=(15, 5))

        # Scatter plot
        axes[0].scatter(x, y, alpha=0.5)
        z = np.polyfit(x, y, 1)
        p = np.poly1d(z)
        axes[0].plot(x.sort_values(), p(x.sort_values()), "r--", linewidth=2)
        axes[0].set_xlabel(var1)
        axes[0].set_ylabel(var2)
        axes[0].set_title(f'Scatter Plot (r={pearson_r:.3f})')

        # Hexbin
        axes[1].hexbin(x, y, gridsize=20, cmap='YlOrRd')
        axes[1].set_xlabel(var1)
        axes[1].set_ylabel(var2)
        axes[1].set_title('Densité')

        # Régression avec IC
        from scipy.stats import linregress
        slope, intercept, r, p, se = linregress(x, y)
        axes[2].scatter(x, y, alpha=0.5)
        x_line = np.linspace(x.min(), x.max(), 100)
        y_line = slope * x_line + intercept
        axes[2].plot(x_line, y_line, 'r-', linewidth=2, label=f'y = {slope:.2f}x + {interc
        axes[2].set_xlabel(var1)
        axes[2].set_ylabel(var2)
        axes[2].set_title('Régression linéaire')
        axes[2].legend()

    elif not x_numeric and not y_numeric:
        # Deux catégorielles: Chi-carré
        contingency = pd.crosstab(x, y)
        chi2, p_value, dof, expected = stats.chi2_contingency(contingency)

        # Cramer's V
        n = len(x)
        min_dim = min(contingency.shape) - 1
        cramers_v = np.sqrt(chi2 / (n * min_dim)) if min_dim > 0 else 0

        print(f"  Test Chi-carré:")
        print(f"    Chi² = {chi2:.4f}")
```

```
        print(f"    p-value = {p_value:.4f}")
        print(f"    Degrés de liberté = {dof}")
        print(f"  V de Cramer = {cramers_v:.4f}")

        if p_value < 0.05:
            print("\n  Interprétation: Association SIGNIFICATIVE entre les variables")
        else:
            print("\n  Interprétation: Pas d'association significative")

        # Visualisation
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))

        contingency.plot(kind='bar', ax=axes[0])
        axes[0].set_title('Tableau croisé')

        sns.heatmap(contingency, annot=True, fmt='d', ax=axes[1], cmap='YlOrRd')
        axes[1].set_title('Heatmap')

    else:
        # Une numérique, une catégorielle: ANOVA ou t-test
        if x_numeric:
            num_var, cat_var = var1, var2
            num_data, cat_data = x, y
        else:
            num_var, cat_var = var2, var1
            num_data, cat_data = y, x

        groups = df.groupby(cat_var)[num_var]
        group_list = [group.dropna().values for name, group in groups]

        if len(group_list) == 2:
            # t-test
            stat, p_value = stats.ttest_ind(group_list[0], group_list[1])
            print(f"  Test t indépendant:")
            print(f"    t = {stat:.4f}")
            print(f"    p-value = {p_value:.4f}")
        else:
            # ANOVA
            stat, p_value = stats.f_oneway(*group_list)
            print(f"  ANOVA (F-test):")
            print(f"    F = {stat:.4f}")
            print(f"    p-value = {p_value:.4f}")

        # Effect size (eta-squared)
        ss_between = sum(len(g) * (g.mean() - num_data.mean())**2 for g in group_list)
        ss_total = sum((num_data - num_data.mean())**2)
        eta_squared = ss_between / ss_total
        print(f"  Eta² (taille d'effet) = {eta_squared:.4f}")

        if p_value < 0.05:
            print("\n  Interprétation: Différence SIGNIFICATIVE entre les groupes")
        else:
            print("\n  Interprétation: Pas de différence significative")
```

```
        # Visualisation
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))

        df.boxplot(column=num_var, by=cat_var, ax=axes[0])
        axes[0].set_title(f'{num_var} par {cat_var}')

        for name, group in groups:
            group.plot.kde(ax=axes[1], label=name)
        axes[1].legend()
        axes[1].set_title('Distributions par groupe')

    plt.tight_layout()
    plt.savefig(f'bivariee_{var1}_{var2}.png', dpi=300, bbox_inches='tight')
    plt.show()

# Application bancaire
np.random.seed(42)
df_bivariee = pd.DataFrame({
    'revenu': np.random.lognormal(10, 0.5, 1000),
    'score_credit': np.random.randint(300, 850, 1000),
    'defaut': np.random.choice([0, 1], 1000, p=[0.95, 0.05]),
    'type_emploi': np.random.choice(['CDI', 'CDD', 'Independant'], 1000, p=[0.6, 0.25, 0.1
})
df_bivariee['score_credit'] = df_bivariee['score_credit'] - df_bivariee['defaut'] * 100

analyse_bivariee(df_bivariee, 'revenu', 'score_credit')
analyse_bivariee(df_bivariee, 'type_emploi', 'score_credit')
```

---

4. Analyse Multivariée

4.1 Régression Multiple

```
def regression_multiple_bancaire(df, target, features):
    """
    Régression multiple complète pour prévision bancaire
    """
    import statsmodels.api as sm
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

    print("=" * 70)
    print(f"RÉGRESSION MULTIPLE: Prédiction de {target}")
    print("=" * 70)

    # Préparer les données
    df_clean = df[features + [target]].dropna()
    X = df_clean[features]
    y = df_clean[target]

    print(f"\n1. DONNÉES")
    print("-" * 40)
```

```
print(f"  Observations: {len(df_clean)}")
print(f"  Features: {features}")
print(f"  Target: {target}")

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

# Standardiser
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 2. Modèle statsmodels pour l'interprétation
print("\n2. RÉSULTATS DU MODÈLE")
print("-" * 40)

X_train_sm = sm.add_constant(X_train)
model_sm = sm.OLS(y_train, X_train_sm).fit()
print(model_sm.summary())

# 3. Diagnostics
print("\n3. DIAGNOSTICS")
print("-" * 40)

# Prédictions
X_test_sm = sm.add_constant(X_test)
y_pred = model_sm.predict(X_test_sm)

# Métriques
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"  RMSE: {rmse:.4f}")
print(f"  MAE: {mae:.4f}")
print(f"  R²: {r2:.4f}")
print(f"  R² ajusté: {model_sm.rsquared_adj:.4f}")

# Visualisation des diagnostics
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Résidus vs Fitted
residuals = y_test - y_pred
axes[0, 0].scatter(y_pred, residuals, alpha=0.5)
axes[0, 0].axhline(0, color='red', linestyle='--')
axes[0, 0].set_xlabel('Valeurs prédites')
axes[0, 0].set_ylabel('Résidus')
axes[0, 0].set_title('Résidus vs Fitted')

# QQ Plot des résidus
stats.probplot(residuals, dist="norm", plot=axes[0, 1])
```

```
    axes[0, 1].set_title('QQ Plot des résidus')

    # Histogramme des résidus
    axes[1, 0].hist(residuals, bins=30, edgecolor='black', density=True)
    axes[1, 0].set_xlabel('Résidus')
    axes[1, 0].set_title('Distribution des résidus')

    # Prédit vs Observé
    axes[1, 1].scatter(y_test, y_pred, alpha=0.5)
    axes[1, 1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
                    'r--', linewidth=2)
    axes[1, 1].set_xlabel('Valeurs observées')
    axes[1, 1].set_ylabel('Valeurs prédites')
    axes[1, 1].set_title('Prédit vs Observé')

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

    # 4. Importance des variables
    print("\n4. IMPORTANCE DES VARIABLES")
    print("-" * 40)

    # Coefficients standardisés
    coefs = pd.DataFrame({
        'Variable': features,
        'Coefficient': model_sm.params[1:],  # Exclure constante
        'p-value': model_sm.pvalues[1:],
        'Significatif': model_sm.pvalues[1:] < 0.05
    }).sort_values('Coefficient', key=abs, ascending=False)

    print(coefs.to_string(index=False))

    return model_sm, coefs

# Application: Prédire le montant de prêt accordé
np.random.seed(42)
n = 1000

df_pret = pd.DataFrame({
    'revenu_mensuel': np.random.lognormal(10, 0.5, n),
    'anciennete_emploi': np.random.exponential(5, n),
    'score_credit': np.random.randint(400, 850, n),
    'nb_credits_actifs': np.random.poisson(1.5, n),
    'age': np.random.randint(25, 65, n)
})

# Créer la cible (montant accordé) avec une relation réaliste
df_pret['montant_accorde'] = (
    50000 +
    0.3 * df_pret['revenu_mensuel'] * 12 +  # 30% du revenu annuel
    1000 * df_pret['anciennete_emploi'] +
    50 * df_pret['score_credit'] +
    -5000 * df_pret['nb_credits_actifs'] +
```

```
    np.random.normal(0, 20000, n)
)
df_pret['montant_accorde'] = df_pret['montant_accorde'].clip(0)

model, importance = regression_multiple_bancaire(
    df_pret,
    target='montant_accorde',
    features=['revenu_mensuel', 'anciennete_emploi', 'score_credit', 'nb_credits_actifs',
)
```

4.2 Prévision Multivariée avec Séries Temporelles

```
def prevision_multivariee_var(df, variables, horizon=6):
    """
    Modèle VAR (Vector AutoRegression) pour prévision multivariée

    Exemple: Prévoir simultanément dépôts, prêts et revenus
    """
    from statsmodels.tsa.api import VAR
    from statsmodels.tsa.stattools import adfuller, grangercausalitytests

    print("=" * 70)
    print("PRÉVISION MULTIVARIÉE (VAR)")
    print("=" * 70)

    # 1. Préparer les données
    df_var = df[variables].copy()

    # 2. Test de stationnarité
    print("\n1. TESTS DE STATIONNARITÉ (ADF)")
    print("-" * 40)

    stationary_results = {}
    for col in variables:
        result = adfuller(df_var[col].dropna())
        stationary_results[col] = {
            'ADF Statistic': result[0],
            'p-value': result[1],
            'Stationnaire': result[1] < 0.05
        }
        print(f"  {col}: ADF={result[0]:.4f}, p={result[1]:.4f} " +
              f"({'Stationnaire' if result[1] < 0.05 else 'Non stationnaire'})")

    # Différencier si nécessaire
    df_diff = df_var.diff().dropna()

    # 3. Sélection de l'ordre (lag)
    print("\n2. SÉLECTION DE L'ORDRE")
    print("-" * 40)

    model = VAR(df_diff)
    lag_order = model.select_order(maxlags=12)
    print(lag_order.summary())
```

```python
optimal_lag = lag_order.aic
print(f"\n  Lag optimal (AIC): {optimal_lag}")

# 4. Estimer le modèle
print("\n3. ESTIMATION DU MODÈLE VAR")
print("-" * 40)

results = model.fit(optimal_lag)
print(results.summary())

# 5. Causalité de Granger
print("\n4. TESTS DE CAUSALITÉ DE GRANGER")
print("-" * 40)

for i, col1 in enumerate(variables):
    for j, col2 in enumerate(variables):
        if i != j:
            test_result = grangercausalitytests(
                df_diff[[col2, col1]].dropna(),
                maxlag=optimal_lag,
                verbose=False
            )
            min_p = min([test_result[lag+1][0]['ssr_ftest'][1] for lag in range(optima
            if min_p < 0.05:
                print(f"  {col1} Granger-cause {col2} (p={min_p:.4f})")

# 6. Prévisions
print("\n5. PRÉVISIONS")
print("-" * 40)

forecast = results.forecast(df_diff.values[-optimal_lag:], steps=horizon)
forecast_df = pd.DataFrame(forecast, columns=variables)

# Reconvertir (cumuler les différences)
last_values = df_var.iloc[-1]
forecast_cumsum = forecast_df.cumsum()
forecast_final = forecast_cumsum + last_values.values

print(f"\nPrévisions sur {horizon} périodes:")
print(forecast_final)

# Visualisation
fig, axes = plt.subplots(len(variables), 1, figsize=(12, 4*len(variables)))

for i, col in enumerate(variables):
    ax = axes[i] if len(variables) > 1 else axes

    # Historique
    df_var[col].plot(ax=ax, label='Historique', linewidth=2)

    # Prévisions
    forecast_idx = pd.date_range(
        df_var.index[-1],
        periods=horizon+1,
```

```
            freq=df_var.index.freq
        )[1:]
        ax.plot(forecast_idx, forecast_final[col],
                'r--', linewidth=2, label='Prévision')

        ax.legend()
        ax.set_title(f'Prévision: {col}')

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

    return results, forecast_final

# Exemple: Prévision conjointe dépôts/prêts/revenus d'intérêts
dates = pd.date_range('2020-01-01', periods=48, freq='M')
np.random.seed(42)

df_bank = pd.DataFrame({
    'depots': 1000 + 20 * np.arange(48) + 50 * np.sin(np.arange(48) * np.pi / 6) + np.rand
    'prets': 800 + 15 * np.arange(48) + 30 * np.sin(np.arange(48) * np.pi / 6) + np.random
    'revenus_interets': 50 + 1 * np.arange(48) + 5 * np.sin(np.arange(48) * np.pi / 6) + n
}, index=dates)

model_var, previsions = prevision_multivariee_var(
    df_bank,
    ['depots', 'prets', 'revenus_interets'],
    horizon=6
)
```

---

5. Application Complète: Scoring de Crédit

5.1 Pipeline Complet

```
class AnalysePrevisionBancaire:
    """
    Pipeline complet d'analyse et prévision pour le scoring bancaire
    """

    def __init__(self, df, target):
        self.df = df.copy()
        self.target = target
        self.resultats = {}

    def analyse_univariee_toutes_variables(self):
        """Étape 1: Analyse univariée de toutes les variables"""
        print("\n" + "=" * 70)
        print("ÉTAPE 1: ANALYSE UNIVARIÉE")
        print("=" * 70)

        for col in self.df.columns:
            if col == self.target:
```

```
            continue

        print(f"\n--- {col} ---")

        if self.df[col].dtype in ['int64', 'float64']:
            stats = {
                'mean': self.df[col].mean(),
                'median': self.df[col].median(),
                'std': self.df[col].std(),
                'skew': self.df[col].skew(),
                'missing': self.df[col].isnull().sum()
            }
            print(f"  Moyenne: {stats['mean']:.2f}")
            print(f"  Médiane: {stats['median']:.2f}")
            print(f"  Skewness: {stats['skew']:.3f}")

            self.resultats[f'univariee_{col}'] = stats

def analyse_bivariee_target(self):
    """Étape 2: Relation de chaque variable avec la cible"""
    print("\n" + "=" * 70)
    print("ÉTAPE 2: ANALYSE BIVARIÉE AVEC LA CIBLE")
    print("=" * 70)

    correlations = {}

    for col in self.df.columns:
        if col == self.target:
            continue

        if self.df[col].dtype in ['int64', 'float64']:
            corr, p_value = stats.pointbiserialr(
                self.df[self.target].dropna(),
                self.df[col].dropna()
            ) if self.df[self.target].nunique() == 2 else stats.pearsonr(
                self.df[self.target].dropna(),
                self.df[col].dropna()
            )

            correlations[col] = {'correlation': corr, 'p_value': p_value}
            print(f"  {col}: r = {corr:.4f} (p = {p_value:.4f})")

    self.resultats['correlations_target'] = correlations

    return pd.DataFrame(correlations).T.sort_values('correlation', key=abs, ascending=

def matrice_correlation(self):
    """Étape 3: Matrice de corrélation multivariée"""
    print("\n" + "=" * 70)
    print("ÉTAPE 3: MATRICE DE CORRÉLATION")
    print("=" * 70)

    numeric_cols = self.df.select_dtypes(include=[np.number]).columns
    corr_matrix = self.df[numeric_cols].corr()
```

```
        # Visualisation
        plt.figure(figsize=(12, 10))
        mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
        sns.heatmap(corr_matrix, mask=mask, annot=True, fmt='.2f',
                    cmap='coolwarm', center=0, square=True)
        plt.title('Matrice de Corrélation')
        plt.tight_layout()
        plt.savefig('matrice_correlation.png', dpi=300)
        plt.show()

        # Identifier les corrélations fortes (multicolinéarité)
        print("\nCorrélations fortes (|r| > 0.7):")
        for i in range(len(corr_matrix.columns)):
            for j in range(i+1, len(corr_matrix.columns)):
                if abs(corr_matrix.iloc[i, j]) > 0.7:
                    print(f"  {corr_matrix.columns[i]} - {corr_matrix.columns[j]}: {corr_m

        self.resultats['correlation_matrix'] = corr_matrix
        return corr_matrix

    def construire_modele(self, features):
        """Étape 4: Construire le modèle de prévision"""
        print("\n" + "=" * 70)
        print("ÉTAPE 4: MODÈLE DE PRÉVISION")
        print("=" * 70)

        from sklearn.model_selection import cross_val_score
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.preprocessing import StandardScaler

        # Préparer les données
        X = self.df[features].dropna()
        y = self.df.loc[X.index, self.target]

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

        # Comparer les modèles
        modeles = {
            'Logistic Regression': LogisticRegression(max_iter=1000),
            'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
        }

        resultats_modeles = {}
        for nom, modele in modeles.items():
            scores = cross_val_score(modele, X_scaled, y, cv=5, scoring='roc_auc')
            resultats_modeles[nom] = {
                'AUC_mean': scores.mean(),
                'AUC_std': scores.std()
            }
            print(f"\n{nom}:")
```

```
            print(f"  AUC: {scores.mean():.4f} (+/- {scores.std():.4f})")

        self.resultats['modeles'] = resultats_modeles
        return resultats_modeles

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

df_scoring = pd.DataFrame({
    'age': np.random.randint(18, 70, n),
    'revenu': np.random.lognormal(10, 0.5, n),
    'anciennete': np.random.exponential(5, n),
    'nb_credits': np.random.poisson(1.5, n),
    'score_bureau': np.random.randint(300, 850, n),
    'ratio_dette': np.random.uniform(0, 0.6, n),
    'defaut': np.random.choice([0, 1], n, p=[0.93, 0.07])
})

# Pipeline d'analyse
analyseur = AnalysePrevisionBancaire(df_scoring, target='defaut')
analyseur.analyse_univariee_toutes_variables()
correlations = analyseur.analyse_bivariee_target()
matrice = analyseur.matrice_correlation()
modeles = analyseur.construire_modele(['age', 'revenu', 'anciennete', 'nb_credits', 'score
```

---

6. Résumé et Bonnes Pratiques

6.1 Workflow Recommandé

```
1. ANALYSE UNIVARIÉE
   │
   ├── Statistiques descriptives
   ├── Détection des outliers
   ├── Test de normalité
   └── Traitement des manquants
   │
   ▼
2. ANALYSE BIVARIÉE
   │
   ├── Corrélations (Pearson/Spearman)
   ├── Tests (t-test, Chi², ANOVA)
   └── Visualisations
   │
   ▼
3. ANALYSE MULTIVARIÉE
   │
   ├── Matrice de corrélation
   ├── VIF (multicolinéarité)
   ├── ACP si nécessaire
   └── Modélisation
   │
```

```
   ▼
4. PRÉVISION
   │
   ├── Validation croisée
   ├── Métriques appropriées
   └── Interprétation
```

6.2 Mnémotechniques

“UBM” pour les types d’analyse: - **U**nivariée = UNE variable - **B**ivariée = DEUX variables - **M**ultivariée = MULTIPLES variables

Choix du test bivariée: “2NC” - **2** Numériques → Corrélation - **N**umérique + Catégorielle → t-test/ANOVA - **2** Catégorielles → Chi-carré

6.3 Erreurs à Éviter

Erreur	Conséquence	Solution
Ignorer la non-normalité	Corrélation de Pearson biaisée	Utiliser Spearman
Multicolinéarité	Coefficients instables	VIF, ACP, sélection
Overfitting	Mauvaise généralisation	Validation croisée
Ignorer les outliers	Résultats biaisés	Analyse préalable