Manuel de Préparation: Analyse Univariée et Multivariée

Introduction

L'analyse univariée examine une variable à la fois, tandis que l'analyse multivariée étudie simultanément plusieurs variables et leurs interrelations. Ces techniques sont fondamentales pour comprendre les données et extraire des insights dans le contexte bancaire.

Partie 1: Analyse Univariée

1.1 Définition et Objectifs

Analyse univariée: Étude d'une seule variable à la fois.

Objectifs: - Décrire la distribution de chaque variable - Identifier les valeurs centrales et la dispersion - Détecter les anomalies et outliers - Préparer l'analyse multivariée

---

1.2 Analyse Univariée - Variables Quantitatives

Mesures Numériques

```
import pandas as pd
import numpy as np
from scipy import stats

def univariate_analysis_numeric(series):
    """Analyse univariée complète d'une variable numérique"""

    analysis = {
        # Tendance centrale
        'mean': series.mean(),
        'median': series.median(),
        'mode': series.mode().iloc[0] if not series.mode().empty else None,

        # Dispersion
        'std': series.std(),
        'var': series.var(),
        'range': series.max() - series.min(),
        'iqr': series.quantile(0.75) - series.quantile(0.25),
        'cv': series.std() / series.mean() * 100,

        # Position
        'min': series.min(),
        'Q1': series.quantile(0.25),
        'Q3': series.quantile(0.75),
        'max': series.max(),
        'percentiles': series.quantile([0.05, 0.1, 0.9, 0.95, 0.99]).to_dict(),

        # Forme
        'skewness': series.skew(),
        'kurtosis': series.kurtosis(),
```

```
        # Qualité
        'count': len(series),
        'missing': series.isnull().sum(),
        'missing_pct': series.isnull().mean() * 100,

        # Outliers (méthode IQR)
        'outliers_count': len(series[(series < series.quantile(0.25) - 1.5 * (series.quantile(0.75) - s
                                     (series > series.quantile(0.75) + 1.5 * (series.quantile(0.75) - s
    }

    return analysis
```

Visualisations

```
import matplotlib.pyplot as plt
import seaborn as sns

def plot_univariate_numeric(series, title="Distribution"):
    """Visualisations pour variable numérique"""

    fig, axes = plt.subplots(2, 2, figsize=(12, 10))

    # Histogramme
    axes[0, 0].hist(series.dropna(), bins=30, edgecolor='black', alpha=0.7)
    axes[0, 0].axvline(series.mean(), color='red', linestyle='--', label=f'Mean: {series.mean():.2f}')
    axes[0, 0].axvline(series.median(), color='green', linestyle='--', label=f'Median: {series.median()
    axes[0, 0].set_title('Histogramme')
    axes[0, 0].legend()

    # Box plot
    axes[0, 1].boxplot(series.dropna())
    axes[0, 1].set_title('Box Plot')

    # Density plot (KDE)
    series.dropna().plot(kind='kde', ax=axes[1, 0])
    axes[1, 0].set_title('Densité (KDE)')

    # QQ plot (normalité)
    stats.probplot(series.dropna(), dist="norm", plot=axes[1, 1])
    axes[1, 1].set_title('QQ Plot')

    plt.suptitle(title)
    plt.tight_layout()
    plt.show()
```

Test de Normalité

```
from scipy.stats import shapiro, normaltest, kstest

def test_normality(series, alpha=0.05):
    """Tests de normalité"""

    data = series.dropna()
    results = {}
```

```
    # Shapiro-Wilk (n < 5000)
    if len(data) < 5000:
        stat, p = shapiro(data)
        results['shapiro'] = {'statistic': stat, 'p_value': p, 'normal': p > alpha}

    # D'Agostino-Pearson (n >= 20)
    if len(data) >= 20:
        stat, p = normaltest(data)
        results['dagostino'] = {'statistic': stat, 'p_value': p, 'normal': p > alpha}

    # Kolmogorov-Smirnov
    stat, p = kstest(data, 'norm', args=(data.mean(), data.std()))
    results['ks'] = {'statistic': stat, 'p_value': p, 'normal': p > alpha}

    return results
```

1.3 Analyse Univariée - Variables Qualitatives

Mesures de Fréquence

```
def univariate_analysis_categorical(series):
    """Analyse univariée d'une variable catégorielle"""

    freq_abs = series.value_counts()
    freq_rel = series.value_counts(normalize=True)

    analysis = {
        'count': len(series),
        'unique': series.nunique(),
        'missing': series.isnull().sum(),
        'missing_pct': series.isnull().mean() * 100,
        'mode': series.mode().iloc[0] if not series.mode().empty else None,
        'mode_freq': freq_rel.iloc[0] if len(freq_rel) > 0 else None,
        'distribution': pd.DataFrame({
            'Effectif': freq_abs,
            'Fréquence': freq_rel,
            'Pourcentage': freq_rel * 100,
            'Cumul': freq_rel.cumsum() * 100
        })
    }

    return analysis
```

Visualisations

```
def plot_univariate_categorical(series, title="Distribution", top_n=10):
    """Visualisations pour variable catégorielle"""

    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Bar chart
    counts = series.value_counts().head(top_n)
```

```
    counts.plot(kind='bar', ax=axes[0], edgecolor='black')
    axes[0].set_title('Diagramme en barres')
    axes[0].set_ylabel('Effectif')
    plt.setp(axes[0].xaxis.get_majorticklabels(), rotation=45, ha='right')

    # Pie chart (si peu de catégories)
    if series.nunique() <= 6:
        series.value_counts().plot(kind='pie', ax=axes[1], autopct='%1.1f%%')
        axes[1].set_title('Diagramme circulaire')
        axes[1].set_ylabel('')
    else:
        # Sinon horizontal bar
        counts.plot(kind='barh', ax=axes[1])
        axes[1].set_title('Top catégories')

    plt.suptitle(title)
    plt.tight_layout()
    plt.show()
```

1.4 Cas Pratique: Analyse Univariée Bancaire

```
def banking_univariate_report(df):
    """Rapport d'analyse univariée pour données bancaires"""

    report = {}

    # Variables numériques typiques
    numeric_vars = ['solde', 'montant_transaction', 'revenu', 'age', 'anciennete']
    for var in numeric_vars:
        if var in df.columns:
            report[var] = univariate_analysis_numeric(df[var])

    # Variables catégorielles typiques
    categorical_vars = ['segment', 'type_compte', 'agence', 'canal', 'produit']
    for var in categorical_vars:
        if var in df.columns:
            report[var] = univariate_analysis_categorical(df[var])

    return report
```

Partie 2: Analyse Bivariée

2.1 Définition et Types

Analyse bivariée: Étude de la relation entre deux variables.

Variable 1	Variable 2	Technique
Quantitative	Quantitative	Corrélation, Scatter plot
Quantitative	Qualitative	Comparaison de moyennes, Box plot
Qualitative	Qualitative	Tableau de contingence, Chi-carré

2.2 Quantitative vs Quantitative

Corrélation

```python
def analyze_correlation(df, var1, var2):
    """Analyse de corrélation entre deux variables numériques"""

    from scipy.stats import pearsonr, spearmanr

    # Suppression des valeurs manquantes
    data = df[[var1, var2]].dropna()

    # Pearson (linéaire)
    r_pearson, p_pearson = pearsonr(data[var1], data[var2])

    # Spearman (monotone)
    r_spearman, p_spearman = spearmanr(data[var1], data[var2])

    return {
        'n': len(data),
        'pearson_r': r_pearson,
        'pearson_p': p_pearson,
        'spearman_rho': r_spearman,
        'spearman_p': p_spearman,
        'interpretation': interpret_correlation(r_pearson)
    }

def interpret_correlation(r):
    """Interprétation du coefficient de corrélation"""
    abs_r = abs(r)
    direction = "positive" if r > 0 else "négative"

    if abs_r < 0.1:
        strength = "négligeable"
    elif abs_r < 0.3:
        strength = "faible"
    elif abs_r < 0.5:
        strength = "modérée"
    elif abs_r < 0.7:
        strength = "substantielle"
    else:
        strength = "forte"

    return f"Corrélation {strength} {direction}"
```

Régression Linéaire Simple

```python
from scipy.stats import linregress

def simple_linear_regression(x, y):
    """Régression linéaire simple"""
```

```
    slope, intercept, r_value, p_value, std_err = linregress(x, y)

    return {
        'slope': slope,
        'intercept': intercept,
        'r_squared': r_value ** 2,
        'p_value': p_value,
        'std_error': std_err,
        'equation': f"y = {slope:.4f}x + {intercept:.4f}"
    }
```

Visualisation

```
def plot_bivariate_numeric(df, var1, var2):
    """Visualisation relation entre deux variables numériques"""

    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Scatter plot avec régression
    axes[0].scatter(df[var1], df[var2], alpha=0.5)

    # Ligne de régression
    z = np.polyfit(df[var1].dropna(), df[var2].dropna(), 1)
    p = np.poly1d(z)
    axes[0].plot(df[var1].sort_values(), p(df[var1].sort_values()),
                 "r--", label=f'y = {z[0]:.2f}x + {z[1]:.2f}')

    axes[0].set_xlabel(var1)
    axes[0].set_ylabel(var2)
    axes[0].set_title('Scatter Plot avec Régression')
    axes[0].legend()

    # Heatmap hexbin (pour grandes données)
    axes[1].hexbin(df[var1], df[var2], gridsize=30, cmap='YlOrRd')
    axes[1].set_xlabel(var1)
    axes[1].set_ylabel(var2)
    axes[1].set_title('Hexbin Plot')

    plt.tight_layout()
    plt.show()
```

2.3 Quantitative vs Qualitative

Comparaison de Moyennes

```
def analyze_numeric_by_category(df, numeric_var, categorical_var):
    """Analyse d'une variable numérique par catégorie"""

    from scipy.stats import ttest_ind, f_oneway, mannwhitneyu, kruskal

    # Statistiques par groupe
    stats_by_group = df.groupby(categorical_var)[numeric_var].agg([
        'count', 'mean', 'median', 'std', 'min', 'max'
```

```
    ])

    # Préparation des groupes
    groups = [group[numeric_var].dropna().values
              for name, group in df.groupby(categorical_var)]

    # Tests statistiques
    n_groups = len(groups)

    if n_groups == 2:
        # t-test ou Mann-Whitney
        t_stat, t_p = ttest_ind(groups[0], groups[1])
        u_stat, u_p = mannwhitneyu(groups[0], groups[1])
        test_results = {
            't_test': {'statistic': t_stat, 'p_value': t_p},
            'mann_whitney': {'statistic': u_stat, 'p_value': u_p}
        }
    else:
        # ANOVA ou Kruskal-Wallis
        f_stat, f_p = f_oneway(*groups)
        h_stat, h_p = kruskal(*groups)
        test_results = {
            'anova': {'statistic': f_stat, 'p_value': f_p},
            'kruskal': {'statistic': h_stat, 'p_value': h_p}
        }

    # Taille de l'effet (Eta-squared pour ANOVA)
    if n_groups > 2:
        grand_mean = df[numeric_var].mean()
        ss_between = sum(len(g) * (np.mean(g) - grand_mean)**2 for g in groups)
        ss_total = sum((df[numeric_var] - grand_mean)**2)
        eta_squared = ss_between / ss_total
        test_results['eta_squared'] = eta_squared

    return {
        'stats_by_group': stats_by_group,
        'test_results': test_results
    }
```

Visualisation

```
def plot_numeric_by_category(df, numeric_var, categorical_var):
    """Visualisation numérique par catégorie"""

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

    # Box plot
    df.boxplot(column=numeric_var, by=categorical_var, ax=axes[0])
    axes[0].set_title('Box Plot')

    # Violin plot
    categories = df[categorical_var].unique()
    data = [df[df[categorical_var] == cat][numeric_var].dropna() for cat in categories]
    parts = axes[1].violinplot(data, showmeans=True, showmedians=True)
```

```
    axes[1].set_xticks(range(1, len(categories) + 1))
    axes[1].set_xticklabels(categories, rotation=45)
    axes[1].set_title('Violin Plot')

    # Bar plot des moyennes avec erreur standard
    means = df.groupby(categorical_var)[numeric_var].mean()
    stds = df.groupby(categorical_var)[numeric_var].std()
    counts = df.groupby(categorical_var)[numeric_var].count()
    se = stds / np.sqrt(counts)

    axes[2].bar(means.index, means.values, yerr=se.values, capsize=5)
    axes[2].set_ylabel(f'Moyenne de {numeric_var}')
    axes[2].set_title('Moyennes avec Erreur Standard')
    plt.setp(axes[2].xaxis.get_majorticklabels(), rotation=45)

    plt.tight_layout()
    plt.show()
```

2.4 Qualitative vs Qualitative

Tableau de Contingence et Chi-Carré

```
def analyze_categorical_association(df, var1, var2):
    """Analyse d'association entre deux variables catégorielles"""

    from scipy.stats import chi2_contingency

    # Tableau de contingence
    contingency = pd.crosstab(df[var1], df[var2])
    contingency_pct_row = pd.crosstab(df[var1], df[var2], normalize='index') * 100
    contingency_pct_col = pd.crosstab(df[var1], df[var2], normalize='columns') * 100

    # Test Chi-carré
    chi2, p_value, dof, expected = chi2_contingency(contingency)

    # V de Cramer
    n = contingency.sum().sum()
    min_dim = min(contingency.shape) - 1
    cramers_v = np.sqrt(chi2 / (n * min_dim))

    return {
        'contingency': contingency,
        'contingency_pct_row': contingency_pct_row,
        'contingency_pct_col': contingency_pct_col,
        'expected': pd.DataFrame(expected, index=contingency.index, columns=contingency.columns),
        'chi2': chi2,
        'p_value': p_value,
        'dof': dof,
        'cramers_v': cramers_v,
        'interpretation': interpret_cramers_v(cramers_v)
    }

def interpret_cramers_v(v):
```

```
    """Interprétation du V de Cramer"""
    if v < 0.1:
        return "Association négligeable"
    elif v < 0.2:
        return "Association faible"
    elif v < 0.4:
        return "Association modérée"
    elif v < 0.6:
        return "Association relativement forte"
    else:
        return "Association forte"
```

Visualisation

```
def plot_categorical_association(df, var1, var2):
    """Visualisation association catégorielle"""

    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Heatmap
    contingency = pd.crosstab(df[var1], df[var2])
    sns.heatmap(contingency, annot=True, fmt='d', cmap='YlOrRd', ax=axes[0])
    axes[0].set_title('Tableau de Contingence')

    # Stacked bar chart
    contingency_pct = pd.crosstab(df[var1], df[var2], normalize='index')
    contingency_pct.plot(kind='bar', stacked=True, ax=axes[1])
    axes[1].set_title('Distribution par Catégorie')
    axes[1].legend(title=var2, bbox_to_anchor=(1.05, 1))

    plt.tight_layout()
    plt.show()
```

Partie 3: Analyse Multivariée

3.1 Vue d'Ensemble

Analyse multivariée: Étude simultanée de 3+ variables et leurs interrelations.

Technique	Objectif	Variables
Matrice de corrélation	Relations linéaires	Toutes numériques
PCA	Réduction de dimensionnalité	Numériques
Clustering	Segmentation	Mixtes
Régression multiple	Prédiction	Variable cible + prédicteurs
MANOVA	Comparaison multivariée	Numériques par groupes

3.2 Matrice de Corrélation

```
def correlation_matrix_analysis(df, numeric_cols=None):
    """Analyse de la matrice de corrélation"""
```

```
    if numeric_cols is None:
        numeric_cols = df.select_dtypes(include=[np.number]).columns

    corr_matrix = df[numeric_cols].corr()

    # Identifier les fortes corrélations
    high_corr = []
    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:
                high_corr.append({
                    'var1': corr_matrix.columns[i],
                    'var2': corr_matrix.columns[j],
                    'correlation': corr_matrix.iloc[i, j]
                })

    return {
        'correlation_matrix': corr_matrix,
        'high_correlations': high_corr
    }

def plot_correlation_matrix(corr_matrix):
    """Visualisation de la matrice de corrélation"""

    plt.figure(figsize=(12, 10))

    # Masquer la diagonale supérieure
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

    sns.heatmap(corr_matrix, mask=mask, annot=True, fmt='.2f',
                cmap='RdBu_r', center=0, square=True,
                linewidths=0.5, cbar_kws={'shrink': 0.8})

    plt.title('Matrice de Corrélation')
    plt.tight_layout()
    plt.show()
```

3.3 Analyse en Composantes Principales (ACP/PCA)

Concept L'ACP transforme les variables corrélées en composantes orthogonales (non corrélées) ordonnées par variance expliquée.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

def pca_analysis(df, numeric_cols, n_components=None):
    """Analyse en Composantes Principales"""

    # Préparation des données
    data = df[numeric_cols].dropna()

    # Standardisation
```

```
    scaler = StandardScaler()
    data_scaled = scaler.fit_transform(data)

    # PCA
    if n_components is None:
        n_components = min(len(numeric_cols), len(data))

    pca = PCA(n_components=n_components)
    components = pca.fit_transform(data_scaled)

    # Variance expliquée
    explained_variance = pca.explained_variance_ratio_
    cumulative_variance = np.cumsum(explained_variance)

    # Loadings (contributions des variables)
    loadings = pd.DataFrame(
        pca.components_.T,
        columns=[f'PC{i+1}' for i in range(n_components)],
        index=numeric_cols
    )

    return {
        'components': components,
        'explained_variance': explained_variance,
        'cumulative_variance': cumulative_variance,
        'loadings': loadings,
        'n_components_95': np.argmax(cumulative_variance >= 0.95) + 1
    }

def plot_pca_results(pca_results):
    """Visualisation des résultats PCA"""

    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Variance expliquée
    n = len(pca_results['explained_variance'])
    x = range(1, n + 1)

    axes[0].bar(x, pca_results['explained_variance'], alpha=0.7, label='Individuelle')
    axes[0].plot(x, pca_results['cumulative_variance'], 'ro-', label='Cumulative')
    axes[0].axhline(y=0.95, color='g', linestyle='--', label='95%')
    axes[0].set_xlabel('Composante')
    axes[0].set_ylabel('Variance expliquée')
    axes[0].set_title('Scree Plot')
    axes[0].legend()

    # Loadings des 2 premières composantes
    loadings = pca_results['loadings'][['PC1', 'PC2']]
    loadings.plot(kind='bar', ax=axes[1])
    axes[1].set_title('Loadings PC1 et PC2')
    axes[1].axhline(y=0, color='black', linestyle='-')

    plt.tight_layout()
    plt.show()
```

3.4 Clustering (Segmentation)

K-Means

```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

def kmeans_clustering(df, features, n_clusters_range=range(2, 11)):
    """Clustering K-Means avec sélection optimale"""

    # Préparation
    data = df[features].dropna()
    scaler = StandardScaler()
    data_scaled = scaler.fit_transform(data)

    # Méthode du coude et silhouette
    inertias = []
    silhouettes = []

    for k in n_clusters_range:
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        kmeans.fit(data_scaled)
        inertias.append(kmeans.inertia_)
        silhouettes.append(silhouette_score(data_scaled, kmeans.labels_))

    # Meilleur k selon silhouette
    best_k = n_clusters_range[np.argmax(silhouettes)]

    # Clustering final
    final_kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
    labels = final_kmeans.fit_predict(data_scaled)

    # Profil des clusters
    df_clustered = df[features].dropna().copy()
    df_clustered['Cluster'] = labels
    cluster_profiles = df_clustered.groupby('Cluster').mean()

    return {
        'labels': labels,
        'best_k': best_k,
        'inertias': dict(zip(n_clusters_range, inertias)),
        'silhouettes': dict(zip(n_clusters_range, silhouettes)),
        'cluster_profiles': cluster_profiles,
        'cluster_sizes': pd.Series(labels).value_counts().sort_index()
    }
```

Visualisation des Clusters

```python
def plot_clusters(data_scaled, labels, pca_components=None):
    """Visualisation des clusters"""

    fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

```
    # Projection 2D (PCA si nécessaire)
    if pca_components is None:
        pca = PCA(n_components=2)
        pca_components = pca.fit_transform(data_scaled)

    # Scatter plot des clusters
    scatter = axes[0].scatter(pca_components[:, 0], pca_components[:, 1],
                              c=labels, cmap='viridis', alpha=0.6)
    axes[0].set_xlabel('PC1')
    axes[0].set_ylabel('PC2')
    axes[0].set_title('Clusters en 2D')
    plt.colorbar(scatter, ax=axes[0])

    # Distribution par cluster
    pd.Series(labels).value_counts().sort_index().plot(kind='bar', ax=axes[1])
    axes[1].set_xlabel('Cluster')
    axes[1].set_ylabel('Effectif')
    axes[1].set_title('Taille des Clusters')

    plt.tight_layout()
    plt.show()
```

3.5 Régression Multiple

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import statsmodels.api as sm

def multiple_regression(df, target, predictors):
    """Régression linéaire multiple"""

    # Préparation des données
    data = df[[target] + predictors].dropna()
    X = data[predictors]
    y = data[target]

    # Statsmodels pour statistiques détaillées
    X_sm = sm.add_constant(X)
    model_sm = sm.OLS(y, X_sm).fit()

    # Sklearn pour prédictions
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model_sk = LinearRegression()
    model_sk.fit(X_train, y_train)
    y_pred = model_sk.predict(X_test)

    # Métriques
    metrics = {
        'r2_train': model_sk.score(X_train, y_train),
        'r2_test': r2_score(y_test, y_pred),
```

```
        'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
        'mae': mean_absolute_error(y_test, y_pred)
    }

    return {
        'summary': model_sm.summary(),
        'coefficients': pd.DataFrame({
            'Variable': ['Intercept'] + predictors,
            'Coefficient': [model_sm.params[0]] + list(model_sm.params[1:]),
            'Std Error': model_sm.bse.values,
            'p-value': model_sm.pvalues.values
        }),
        'metrics': metrics,
        'model': model_sk
    }
```

3.6 Analyse RFM (Contexte Bancaire)

```
def rfm_analysis(df, customer_id, date_col, amount_col, reference_date=None):
    """Analyse RFM pour segmentation client"""

    if reference_date is None:
        reference_date = df[date_col].max()

    # Calcul RFM
    rfm = df.groupby(customer_id).agg({
        date_col: lambda x: (reference_date - x.max()).days,  # Recency
        customer_id: 'count',  # Frequency (utilise la colonne ID comme compteur)
        amount_col: 'sum'  # Monetary
    })

    rfm.columns = ['Recency', 'Frequency', 'Monetary']

    # Scoring (quintiles)
    rfm['R_Score'] = pd.qcut(rfm['Recency'], 5, labels=[5, 4, 3, 2, 1])  # Inversé: récent = bon
    rfm['F_Score'] = pd.qcut(rfm['Frequency'].rank(method='first'), 5, labels=[1, 2, 3, 4, 5])
    rfm['M_Score'] = pd.qcut(rfm['Monetary'].rank(method='first'), 5, labels=[1, 2, 3, 4, 5])

    rfm['RFM_Score'] = rfm['R_Score'].astype(str) + rfm['F_Score'].astype(str) + rfm['M_Score'].astype(s

    # Segmentation
    def segment_customer(row):
        r, f, m = int(row['R_Score']), int(row['F_Score']), int(row['M_Score'])
        if r >= 4 and f >= 4:
            return 'Champions'
        elif r >= 4 and f <= 2:
            return 'Nouveaux clients'
        elif r <= 2 and f >= 4:
            return 'À risque'
        elif r <= 2 and f <= 2:
            return 'Perdus'
        else:
```

```
            return 'Moyens'

    rfm['Segment'] = rfm.apply(segment_customer, axis=1)

    return rfm
```

Partie 4: Applications Bancaires

4.1 Analyse du Portefeuille Client

```
def portfolio_analysis(df_customers, df_transactions):
    """Analyse multivariée du portefeuille client"""

    # Agrégation des transactions
    tx_summary = df_transactions.groupby('client_id').agg({
        'montant': ['sum', 'mean', 'std', 'count'],
        'date': ['min', 'max']
    })
    tx_summary.columns = ['total', 'moyenne', 'std', 'nb_tx', 'first_tx', 'last_tx']

    # Fusion avec données clients
    df_analysis = df_customers.merge(tx_summary, on='client_id')

    # Variables d'analyse
    numeric_features = ['solde', 'total', 'moyenne', 'nb_tx', 'anciennete', 'nb_produits']

    # Analyses
    results = {
        'correlation': correlation_matrix_analysis(df_analysis, numeric_features),
        'pca': pca_analysis(df_analysis, numeric_features),
        'clustering': kmeans_clustering(df_analysis, numeric_features)
    }

    return results
```

4.2 Analyse de Risque de Crédit

```
def credit_risk_analysis(df_loans):
    """Analyse multivariée du risque de crédit"""

    # Variables
    risk_features = ['montant', 'duree', 'taux', 'revenu', 'dti', 'score_credit', 'anciennete_emploi']
    target = 'defaut'

    # Analyse bivariée: chaque feature vs défaut
    bivariate_results = {}
    for feature in risk_features:
        bivariate_results[feature] = analyze_numeric_by_category(df_loans, feature, target)

    # Corrélations
    corr_analysis = correlation_matrix_analysis(df_loans, risk_features)
```

```python
    # Régression logistique (pour scoring)
    from sklearn.linear_model import LogisticRegression
    X = df_loans[risk_features].dropna()
    y = df_loans.loc[X.index, target]

    model = LogisticRegression()
    model.fit(X, y)

    feature_importance = pd.DataFrame({
        'Feature': risk_features,
        'Coefficient': model.coef_[0],
        'Abs_Coef': np.abs(model.coef_[0])
    }).sort_values('Abs_Coef', ascending=False)

    return {
        'bivariate': bivariate_results,
        'correlations': corr_analysis,
        'feature_importance': feature_importance
    }
```

Résumé

Univariée

- **Objectif:** Comprendre chaque variable individuellement
- **Numériques:** Moyenne, médiane, écart-type, distribution
- **Catégorielles:** Fréquences, mode, proportions

Bivariée

- **Num vs Num:** Corrélation, scatter plot, régression simple
- **Num vs Cat:** Comparaison de moyennes, box plot, ANOVA
- **Cat vs Cat:** Chi-carré, contingence, V de Cramer

Multivariée

- **Réduction:** PCA pour simplifier
- **Segmentation:** Clustering pour grouper
- **Prédiction:** Régression multiple

Questions d'Entretien

1. **Différence entre analyse univariée et multivariée?** → Univariée = 1 variable; Bivariée = 2; Multivariée = 3+
2. **Quand utiliser PCA?** → Pour réduire les dimensions, visualiser, ou éliminer la multicolinéarité
3. **Comment choisir le nombre de clusters?** → Méthode du coude, silhouette score, interprétabilité métier

4. **Qu’est-ce que le V de Cramer?** → Mesure d’association entre variables catégorielles (0 à 1)
5. **Expliquez l’analyse RFM.** → Récence, Fréquence, Montant pour segmenter les clients

---

Rappel: L’analyse multivariée permet de voir les patterns invisibles dans l’analyse univariée. Toujours commencer par l’univariée, puis bivariée, puis multivariée.