ACP et ANOVA: Guide Complet pour Data Analyst Bancaire

1. ANOVA (Analysis of Variance)

1.1 Qu'est-ce que l'ANOVA?

L'ANOVA est un test statistique qui compare les moyennes de **3 groupes ou plus** pour déterminer s'il existe une différence significative entre eux.

Hypothèses: - H_0: $\mu_1 = \mu_2 = \mu_3 = ... = \mu_k$ (toutes les moyennes sont égales) - H_1: Au moins une moyenne est différente

1.2 Types d'ANOVA

```
ANOVA
├── À un facteur (One-way)
│   └── 1 variable catégorielle, 1 variable continue
│       Exemple: Revenu moyen par type d'emploi
│
├── À deux facteurs (Two-way)
│   └── 2 variables catégorielles, 1 variable continue
│       Exemple: Revenu par type d'emploi ET région
│
└── ANOVA à mesures répétées
    └── Mêmes sujets mesurés plusieurs fois
        Exemple: Score de satisfaction avant/après campagne
```

1.3 Conditions d'Application

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

def verifier_conditions_anova(df, variable_continue, variable_groupe):
    """
    Vérifier les 3 conditions de l'ANOVA:
    1. Indépendance des observations
    2. Normalité des distributions par groupe
    3. Homogénéité des variances (homoscédasticité)
    """
    print("=" * 60)
    print("VÉRIFICATION DES CONDITIONS DE L'ANOVA")
    print("=" * 60)

    groupes = df.groupby(variable_groupe)[variable_continue]

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

    # Box plot
    df.boxplot(column=variable_continue, by=variable_groupe, ax=axes[0])
    axes[0].set_title('Distribution par groupe')
```

```python
# Histogrammes superposés
for name, group in groupes:
    group.hist(alpha=0.5, label=name, ax=axes[1], bins=20)
axes[1].legend()
axes[1].set_title('Histogrammes par groupe')

# QQ plots (normalisés)
for i, (name, group) in enumerate(groupes):
    stats.probplot(group.dropna(), dist="norm", plot=axes[2])
axes[2].set_title('QQ-Plot (tous groupes)')

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

# 2. Test de normalité par groupe (Shapiro-Wilk)
print("\n1. NORMALITÉ (Test de Shapiro-Wilk)")
print("-" * 40)

normalite_ok = True
for name, group in groupes:
    if len(group.dropna()) >= 3:
        stat, p_value = stats.shapiro(group.dropna()[:5000])
        status = "OK" if p_value > 0.05 else "NON NORMAL"
        if p_value <= 0.05:
            normalite_ok = False
        print(f"  {name}: W={stat:.4f}, p={p_value:.4f} [{status}]")

# 3. Homogénéité des variances (Test de Levene)
print("\n2. HOMOGÉNÉITÉ DES VARIANCES (Test de Levene)")
print("-" * 40)

groups_list = [group.dropna().values for name, group in groupes]
stat_levene, p_levene = stats.levene(*groups_list)

variance_ok = p_levene > 0.05
print(f"  Statistique: {stat_levene:.4f}")
print(f"  p-value: {p_levene:.4f}")
print(f"  Conclusion: {'Variances homogènes' if variance_ok else 'Variances hétérogène

# 4. Résumé
print("\n3. RÉSUMÉ DES CONDITIONS")
print("-" * 40)

if normalite_ok and variance_ok:
    print("  ✓ Toutes les conditions sont satisfaites → ANOVA paramétrique OK")
    recommandation = "ANOVA paramétrique"
elif not normalite_ok and variance_ok:
    print("  ! Normalité non respectée → Considérer Kruskal-Wallis")
    recommandation = "Kruskal-Wallis"
elif normalite_ok and not variance_ok:
    print("  ! Variances hétérogènes → Utiliser Welch ANOVA")
    recommandation = "Welch ANOVA"
```

```
    else:
        print("  ✗ Conditions non respectées → Utiliser Kruskal-Wallis")
        recommandation = "Kruskal-Wallis"

    return {
        'normalite': normalite_ok,
        'homogeneite_variance': variance_ok,
        'recommandation': recommandation
    }

# Exemple bancaire
np.random.seed(42)
df_emploi = pd.DataFrame({
    'type_emploi': np.random.choice(['CDI', 'CDD', 'Independant', 'Fonctionnaire'], 500,
                                    p=[0.5, 0.2, 0.2, 0.1]),
    'revenu_mensuel': np.random.lognormal(10, 0.5, 500)
})

# Créer des différences entre groupes
df_emploi.loc[df_emploi['type_emploi'] == 'Fonctionnaire', 'revenu_mensuel'] *= 1.3
df_emploi.loc[df_emploi['type_emploi'] == 'CDD', 'revenu_mensuel'] *= 0.8

conditions = verifier_conditions_anova(df_emploi, 'revenu_mensuel', 'type_emploi')
```

1.4 ANOVA à Un Facteur

```
def anova_un_facteur(df, variable_continue, variable_groupe):
    """
    ANOVA à un facteur complète avec post-hoc
    """
    print("=" * 70)
    print(f"ANOVA À UN FACTEUR: {variable_continue} ~ {variable_groupe}")
    print("=" * 70)

    groupes = df.groupby(variable_groupe)[variable_continue]

    # 1. Statistiques descriptives par groupe
    print("\n1. STATISTIQUES PAR GROUPE")
    print("-" * 50)

    stats_groupe = groupes.agg(['count', 'mean', 'std', 'median']).round(2)
    print(stats_groupe)

    # 2. ANOVA
    print("\n2. TEST ANOVA")
    print("-" * 50)

    groups_list = [group.dropna().values for name, group in groupes]
    f_stat, p_value = stats.f_oneway(*groups_list)

    print(f"  F-statistique: {f_stat:.4f}")
    print(f"  p-value: {p_value:.6f}")

    if p_value < 0.05:
```

```
    print("\n  *** RÉSULTAT SIGNIFICATIF ***")
    print("  Il existe une différence significative entre au moins deux groupes")
else:
    print("\n  Pas de différence significative entre les groupes")

# 3. Taille d'effet (Eta-squared)
print("\n3. TAILLE D'EFFET")
print("-" * 50)

# Calculer manuellement
grand_mean = df[variable_continue].mean()
ss_between = sum(len(g) * (g.mean() - grand_mean)**2 for g in groups_list)
ss_total = sum((df[variable_continue] - grand_mean)**2)
eta_squared = ss_between / ss_total

print(f"  η² (Eta-squared): {eta_squared:.4f}")

if eta_squared < 0.01:
    effect_size = "Négligeable"
elif eta_squared < 0.06:
    effect_size = "Petit"
elif eta_squared < 0.14:
    effect_size = "Moyen"
else:
    effect_size = "Grand"

print(f"  Interprétation: Effet {effect_size}")

# 4. Tests post-hoc (si significatif)
if p_value < 0.05:
    print("\n4. TESTS POST-HOC (Tukey HSD)")
    print("-" * 50)

    from scipy.stats import tukey_hsd

    result = tukey_hsd(*groups_list)
    group_names = list(groupes.groups.keys())

    print("\n  Comparaisons deux à deux:")
    for i in range(len(group_names)):
        for j in range(i+1, len(group_names)):
            # Accéder aux p-values
            ci_low = result.confidence_interval().low[i, j]
            ci_high = result.confidence_interval().high[i, j]
            stat = result.statistic[i, j]
            p_adj = result.pvalue[i, j]

            sig = "***" if p_adj < 0.001 else "**" if p_adj < 0.01 else "*" if p_adj <
            print(f"    {group_names[i]} vs {group_names[j]}: diff={stat:.2f}, p={p_ad

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

# Means plot avec IC
```

```
    means = stats_groupe['mean']
    stds = stats_groupe['std']
    counts = stats_groupe['count']
    se = stds / np.sqrt(counts)
    ci95 = 1.96 * se

    x = np.arange(len(means))
    axes[0].bar(x, means, yerr=ci95, capsize=5, alpha=0.7, edgecolor='black')
    axes[0].set_xticks(x)
    axes[0].set_xticklabels(means.index)
    axes[0].set_ylabel(variable_continue)
    axes[0].set_title('Moyennes avec IC 95%')

    # Box plot
    df.boxplot(column=variable_continue, by=variable_groupe, ax=axes[1])
    axes[1].set_title('Distribution par groupe')

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

    return {
        'f_stat': f_stat,
        'p_value': p_value,
        'eta_squared': eta_squared,
        'stats_groupe': stats_groupe
    }

# Application
resultats = anova_un_facteur(df_emploi, 'revenu_mensuel', 'type_emploi')
```

1.5 ANOVA à Deux Facteurs

```
def anova_deux_facteurs(df, variable_continue, facteur1, facteur2):
    """
    ANOVA à deux facteurs avec interaction
    """
    import statsmodels.api as sm
    from statsmodels.formula.api import ols

    print("=" * 70)
    print(f"ANOVA À DEUX FACTEURS: {variable_continue} ~ {facteur1} * {facteur2}")
    print("=" * 70)

    # Créer le modèle avec interaction
    formule = f'{variable_continue} ~ C({facteur1}) + C({facteur2}) + C({facteur1}):C({fac
    model = ols(formule, data=df).fit()

    # Table ANOVA
    anova_table = sm.stats.anova_lm(model, typ=2)

    print("\n1. TABLE ANOVA")
    print("-" * 50)
    print(anova_table)
```

```
    # Interprétation
    print("\n2. INTERPRÉTATION")
    print("-" * 50)

    for effect in anova_table.index[:-1]:  # Exclure Residual
        p_val = anova_table.loc[effect, 'PR(>F)']
        if p_val < 0.05:
            print(f"  {effect}: SIGNIFICATIF (p = {p_val:.4f})")
        else:
            print(f"  {effect}: Non significatif (p = {p_val:.4f})")

    # Visualisation des moyennes marginales
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Graphique d'interaction
    pivot = df.pivot_table(
        values=variable_continue,
        index=facteur1,
        columns=facteur2,
        aggfunc='mean'
    )

    pivot.plot(marker='o', ax=axes[0])
    axes[0].set_ylabel(variable_continue)
    axes[0].set_title('Graphique d\'interaction')
    axes[0].legend(title=facteur2)

    # Heatmap des moyennes
    sns.heatmap(pivot, annot=True, fmt='.0f', cmap='YlOrRd', ax=axes[1])
    axes[1].set_title(f'Moyennes de {variable_continue}')

    plt.tight_layout()
    plt.savefig('anova_2_facteurs.png', dpi=300)
    plt.show()

    return anova_table

# Exemple: Revenu par type d'emploi ET région
np.random.seed(42)
df_2f = pd.DataFrame({
    'type_emploi': np.random.choice(['CDI', 'CDD', 'Independant'], 600),
    'region': np.random.choice(['Nord', 'Sud', 'Ouest'], 600),
    'revenu_mensuel': np.random.lognormal(10, 0.5, 600)
})

# Créer des effets
df_2f.loc[df_2f['region'] == 'Ouest', 'revenu_mensuel'] *= 1.2
df_2f.loc[(df_2f['type_emploi'] == 'CDI') & (df_2f['region'] == 'Ouest'), 'revenu_mensuel'

anova_table = anova_deux_facteurs(df_2f, 'revenu_mensuel', 'type_emploi', 'region')
```

1.6 Application Bancaire: Segmentation des Clients

```
"""
CAS PRATIQUE: Analyser les différences de comportement entre segments de clients
"""

def analyse_segments_bancaires(df):
    """
    Analyse ANOVA des comportements par segment client
    """
    print("=" * 70)
    print("ANALYSE DES SEGMENTS CLIENTS - ANOVA")
    print("=" * 70)

    # Variables à analyser
    variables = {
        'solde_moyen': "Solde moyen du compte",
        'nb_transactions': "Nombre de transactions/mois",
        'montant_moyen_tx': "Montant moyen par transaction",
        'ancienneté_mois': "Ancienneté (mois)"
    }

    resultats = {}

    for var, description in variables.items():
        print(f"\n{'='*50}")
        print(f"Variable: {description}")
        print('='*50)

        # Test ANOVA
        groupes = df.groupby('segment')[var]
        groups_list = [g.dropna().values for _, g in groupes]
        f_stat, p_value = stats.f_oneway(*groups_list)

        # Eta-squared
        grand_mean = df[var].mean()
        ss_between = sum(len(g) * (g.mean() - grand_mean)**2 for g in groups_list)
        ss_total = sum((df[var] - grand_mean)**2)
        eta_sq = ss_between / ss_total

        print(f"  F = {f_stat:.2f}, p = {p_value:.4f}, η² = {eta_sq:.4f}")

        if p_value < 0.05:
            print("  → Différences SIGNIFICATIVES entre segments")

        resultats[var] = {
            'f_stat': f_stat,
            'p_value': p_value,
            'eta_squared': eta_sq,
            'significatif': p_value < 0.05
        }

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

```
    for ax, (var, desc) in zip(axes.flatten(), variables.items()):
        df.boxplot(column=var, by='segment', ax=ax)
        ax.set_title(desc)
        ax.set_xlabel('Segment')

    plt.suptitle('Comparaison des segments clients', fontsize=14, y=1.02)
    plt.tight_layout()
    plt.savefig('segments_anova.png', dpi=300)
    plt.show()

    return pd.DataFrame(resultats).T

# Créer des données de segments
np.random.seed(42)
n = 1000

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

# Créer des différences réalistes
for seg in df_segments['segment'].unique():
    mask = df_segments['segment'] == seg
    n_seg = mask.sum()

    if seg == 'Premium':
        df_segments.loc[mask, 'solde_moyen'] = np.random.lognormal(12, 0.5, n_seg)
        df_segments.loc[mask, 'nb_transactions'] = np.random.poisson(25, n_seg)
        df_segments.loc[mask, 'montant_moyen_tx'] = np.random.lognormal(8, 0.3, n_seg)
        df_segments.loc[mask, 'ancienneté_mois'] = np.random.exponential(60, n_seg) + 24
    elif seg == 'Standard':
        df_segments.loc[mask, 'solde_moyen'] = np.random.lognormal(10, 0.6, n_seg)
        df_segments.loc[mask, 'nb_transactions'] = np.random.poisson(15, n_seg)
        df_segments.loc[mask, 'montant_moyen_tx'] = np.random.lognormal(7, 0.4, n_seg)
        df_segments.loc[mask, 'ancienneté_mois'] = np.random.exponential(36, n_seg) + 12
    else:
        df_segments.loc[mask, 'solde_moyen'] = np.random.lognormal(9, 0.7, n_seg)
        df_segments.loc[mask, 'nb_transactions'] = np.random.poisson(8, n_seg)
        df_segments.loc[mask, 'montant_moyen_tx'] = np.random.lognormal(6, 0.5, n_seg)
        df_segments.loc[mask, 'ancienneté_mois'] = np.random.exponential(18, n_seg) + 3

resultats_segments = analyse_segments_bancaires(df_segments)
print("\nRésumé:")
print(resultats_segments)
```

---

2. ACP (Analyse en Composantes Principales)

2.1 Qu'est-ce que l'ACP?

L'ACP est une technique de **réduction de dimensionnalité** qui transforme des variables corrélées en un ensemble de variables non corrélées appelées **composantes principales**.

Objectifs: 1. Réduire le nombre de dimensions 2. Éliminer la multicolinéarité 3. Visualiser des données multidimensionnelles 4. Identifier les structures latentes

2.2 Quand Utiliser l'ACP?

Situation	ACP appropriée?
Trop de variables corrélées	OUI
Visualiser données HD	OUI
Préparer pour clustering	OUI
Variables catégorielles	NON (utiliser MCA)
Interprétabilité requise	ATTENTION

2.3 ACP Complète avec Interprétation

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

def acp_complete(df, variables, n_components=None):
    """
    ACP complète avec diagnostics et interprétation
    """
    print("=" * 70)
    print("ANALYSE EN COMPOSANTES PRINCIPALES (ACP)")
    print("=" * 70)

    # 1. Préparer les données
    X = df[variables].dropna()

    print(f"\n1. DONNÉES")
    print("-" * 50)
    print(f"  Observations: {len(X)}")
    print(f"  Variables: {len(variables)}")
    print(f"  Variables: {variables}")

    # 2. Standardiser (OBLIGATOIRE pour l'ACP)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # 3. Matrice de corrélation
    print(f"\n2. MATRICE DE CORRÉLATION")
    print("-" * 50)

    corr_matrix = pd.DataFrame(X_scaled, columns=variables).corr()

    plt.figure(figsize=(10, 8))
    sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', center=0)
    plt.title('Matrice de corrélation (données standardisées)')
    plt.tight_layout()
    plt.savefig('acp_correlation.png', dpi=300)
    plt.show()

    # 4. Test KMO (Kaiser-Meyer-Olkin)
```

```python
print(f"\n3. ADÉQUATION DES DONNÉES")
print("-" * 50)

# KMO simplifié
corr_inv = np.linalg.inv(corr_matrix)
partial_corr = -corr_inv / np.sqrt(np.outer(np.diag(corr_inv), np.diag(corr_inv)))
np.fill_diagonal(partial_corr, 0)

kmo_num = (corr_matrix**2).sum().sum() - len(variables)
kmo_denom = kmo_num + (partial_corr**2).sum().sum()
kmo = kmo_num / kmo_denom if kmo_denom != 0 else 0

print(f"  KMO approximatif: {kmo:.4f}")
if kmo >= 0.8:
    print("  Interprétation: Excellent - ACP très appropriée")
elif kmo >= 0.6:
    print("  Interprétation: Acceptable - ACP possible")
else:
    print("  Interprétation: Faible - ACP déconseillée")

# Test de sphéricité de Bartlett
n = len(X)
p = len(variables)
det_corr = np.linalg.det(corr_matrix)
chi2 = -(n - 1 - (2*p + 5)/6) * np.log(det_corr)
df_test = p * (p-1) / 2
p_bartlett = 1 - stats.chi2.cdf(chi2, df_test)

print(f"\n  Test de Bartlett:")
print(f"    Chi² = {chi2:.2f}")
print(f"    p-value = {p_bartlett:.6f}")
if p_bartlett < 0.05:
    print("    → Les variables sont corrélées, ACP appropriée")

# 5. Effectuer l'ACP
print(f"\n4. RÉSULTATS DE L'ACP")
print("-" * 50)

if n_components is None:
    n_components = len(variables)

pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)

# Variance expliquée
var_exp = pca.explained_variance_ratio_
var_cum = np.cumsum(var_exp)

print("\n  Variance expliquée par composante:")
for i, (ve, vc) in enumerate(zip(var_exp, var_cum)):
    print(f"    PC{i+1}: {ve*100:.2f}% (Cumulé: {vc*100:.2f}%)")

# 6. Scree Plot et critère de Kaiser
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

```python
# Scree plot
x = np.arange(1, len(var_exp) + 1)
axes[0].bar(x, var_exp * 100, alpha=0.7, label='Variance individuelle')
axes[0].plot(x, var_cum * 100, 'ro-', label='Variance cumulée')
axes[0].axhline(y=80, color='green', linestyle='--', label='Seuil 80%')
axes[0].set_xlabel('Composante principale')
axes[0].set_ylabel('Variance expliquée (%)')
axes[0].set_title('Scree Plot')
axes[0].legend()
axes[0].set_xticks(x)

# Critère de Kaiser (valeurs propres > 1)
eigenvalues = pca.explained_variance_
axes[1].bar(x, eigenvalues, alpha=0.7)
axes[1].axhline(y=1, color='red', linestyle='--', label='Kaiser (λ = 1)')
axes[1].set_xlabel('Composante principale')
axes[1].set_ylabel('Valeur propre')
axes[1].set_title('Critère de Kaiser')
axes[1].legend()
axes[1].set_xticks(x)

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

# Nombre de composantes recommandé
n_kaiser = (eigenvalues > 1).sum()
n_80pct = (var_cum >= 0.80).argmax() + 1

print(f"\n5. NOMBRE DE COMPOSANTES RECOMMANDÉ")
print("-" * 50)
print(f"  Critère de Kaiser (λ > 1): {n_kaiser} composantes")
print(f"  Critère des 80%: {n_80pct} composantes")

# 7. Loadings (contributions des variables)
print(f"\n6. LOADINGS (Contributions des variables)")
print("-" * 50)

loadings = pd.DataFrame(
    pca.components_.T,
    columns=[f'PC{i+1}' for i in range(n_components)],
    index=variables
)
print(loadings.round(3))

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

# Heatmap des loadings
sns.heatmap(loadings.iloc[:, :min(5, n_components)],
            annot=True, fmt='.2f', cmap='RdBu_r', center=0, ax=ax)
ax.set_title('Loadings des variables sur les composantes principales')
```

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

    # 8. Biplot (si 2D possible)
    if n_components >= 2:
        print(f"\n7. BIPLOT")
        print("-" * 50)

        fig, ax = plt.subplots(figsize=(12, 10))

        # Scores (observations)
        ax.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.3, s=20)

        # Vecteurs (variables)
        scale = max(abs(X_pca[:, 0]).max(), abs(X_pca[:, 1]).max()) / max(abs(loadings['PC

        for i, var in enumerate(variables):
            ax.arrow(0, 0,
                     loadings.loc[var, 'PC1'] * scale * 0.8,
                     loadings.loc[var, 'PC2'] * scale * 0.8,
                     head_width=scale*0.03, head_length=scale*0.02, fc='red', ec='red')
            ax.text(loadings.loc[var, 'PC1'] * scale * 0.9,
                    loadings.loc[var, 'PC2'] * scale * 0.9,
                    var, fontsize=10, color='red')

        ax.axhline(0, color='gray', linestyle='--', alpha=0.5)
        ax.axvline(0, color='gray', linestyle='--', alpha=0.5)
        ax.set_xlabel(f'PC1 ({var_exp[0]*100:.1f}%)')
        ax.set_ylabel(f'PC2 ({var_exp[1]*100:.1f}%)')
        ax.set_title('Biplot ACP')

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

    return {
        'pca': pca,
        'X_pca': X_pca,
        'loadings': loadings,
        'variance_expliquee': var_exp,
        'n_kaiser': n_kaiser,
        'n_80pct': n_80pct,
        'scaler': scaler
    }

# Application bancaire
np.random.seed(42)
n = 1000

# Variables financières corrélées
df_finance = pd.DataFrame({
    'solde_epargne': np.random.lognormal(10, 0.5, n),
    'solde_courant': np.random.lognormal(9, 0.6, n),
```

```
    'revenus_mensuels': np.random.lognormal(10.5, 0.4, n),
    'depenses_mensuelles': np.random.lognormal(10, 0.5, n),
    'montant_credits': np.random.lognormal(11, 0.7, n),
    'nb_transactions': np.random.poisson(20, n).astype(float),
    'anciennete': np.random.exponential(36, n),
    'score_interne': np.random.randint(300, 850, n).astype(float)
})

# Créer des corrélations réalistes
df_finance['depenses_mensuelles'] = 0.7 * df_finance['revenus_mensuels'] + np.random.norma
df_finance['solde_courant'] = 0.3 * df_finance['revenus_mensuels'] - 0.5 * df_finance['dep
df_finance['solde_courant'] = df_finance['solde_courant'].clip(0)

variables_acp = ['solde_epargne', 'solde_courant', 'revenus_mensuels',
                 'depenses_mensuelles', 'montant_credits', 'nb_transactions',
                 'anciennete', 'score_interne']

resultats_acp = acp_complete(df_finance, variables_acp)
```

2.4 Interprétation des Composantes

```
def interpreter_composantes(pca, variables, seuil=0.4):
    """
    Aide à l'interprétation des composantes principales
    """
    print("=" * 70)
    print("INTERPRÉTATION DES COMPOSANTES PRINCIPALES")
    print("=" * 70)

    loadings = pd.DataFrame(
        pca.components_.T,
        columns=[f'PC{i+1}' for i in range(len(pca.components_))],
        index=variables
    )

    for i in range(min(3, len(pca.components_))):
        pc = f'PC{i+1}'
        var_exp = pca.explained_variance_ratio_[i] * 100

        print(f"\n{pc} ({var_exp:.1f}% de variance)")
        print("-" * 50)

        # Variables avec loadings significatifs
        significant = loadings[pc][abs(loadings[pc]) >= seuil].sort_values(ascending=False

        if len(significant) > 0:
            pos = significant[significant > 0]
            neg = significant[significant < 0]

            print("  Contributions POSITIVES:")
            for var, loading in pos.items():
                print(f"    + {var}: {loading:.3f}")

            print("  Contributions NÉGATIVES:")
```

```
            for var, loading in neg.items():
                print(f"    - {var}: {loading:.3f}")

            # Suggestion d'interprétation
            if len(pos) > 0 and len(neg) > 0:
                print(f"\n  Interprétation suggérée:")
                print(f"    Cette composante oppose {list(pos.index)} à {list(neg.index)}"
            elif len(pos) > 0:
                print(f"\n  Interprétation suggérée:")
                print(f"    Cette composante mesure le niveau de {', '.join(pos.index)}")

interpreter_composantes(resultats_acp['pca'], variables_acp)
```

2.5 ACP pour Scoring Bancaire

```
def acp_pour_scoring(df, variables, target, n_components='auto'):
    """
    Utiliser l'ACP comme préparation pour un modèle de scoring
    """
    from sklearn.model_selection import cross_val_score
    from sklearn.linear_model import LogisticRegression
    from sklearn.pipeline import Pipeline

    print("=" * 70)
    print("ACP POUR SCORING DE CRÉDIT")
    print("=" * 70)

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

    # 1. Comparer différents nombres de composantes
    print("\n1. COMPARAISON DES MODÈLES")
    print("-" * 50)

    resultats = []

    # Sans ACP
    pipeline_base = Pipeline([
        ('scaler', StandardScaler()),
        ('model', LogisticRegression(max_iter=1000))
    ])

    scores_base = cross_val_score(pipeline_base, X, y, cv=5, scoring='roc_auc')
    resultats.append({
        'Config': 'Sans ACP',
        'N_features': len(variables),
        'AUC_mean': scores_base.mean(),
        'AUC_std': scores_base.std()
    })
    print(f"  Sans ACP ({len(variables)} features): AUC = {scores_base.mean():.4f}")

    # Avec ACP (différents n_components)
    for n in range(2, min(len(variables), 7)):
```

```
        pipeline_pca = Pipeline([
            ('scaler', StandardScaler()),
            ('pca', PCA(n_components=n)),
            ('model', LogisticRegression(max_iter=1000))
        ])

        scores_pca = cross_val_score(pipeline_pca, X, y, cv=5, scoring='roc_auc')
        resultats.append({
            'Config': f'ACP ({n} comp.)',
            'N_features': n,
            'AUC_mean': scores_pca.mean(),
            'AUC_std': scores_pca.std()
        })
        print(f"  ACP {n} composantes: AUC = {scores_pca.mean():.4f}")

    # Résumé
    df_resultats = pd.DataFrame(resultats)

    # Visualisation
    plt.figure(figsize=(10, 5))
    plt.bar(df_resultats['Config'], df_resultats['AUC_mean'],
            yerr=df_resultats['AUC_std'], capsize=5, alpha=0.7)
    plt.xticks(rotation=45, ha='right')
    plt.ylabel('AUC-ROC')
    plt.title('Impact de l\'ACP sur la performance du scoring')
    plt.tight_layout()
    plt.savefig('acp_scoring_comparison.png', dpi=300)
    plt.show()

    # Recommandation
    best = df_resultats.loc[df_resultats['AUC_mean'].idxmax()]
    print(f"\n2. RECOMMANDATION")
    print("-" * 50)
    print(f"  Meilleure configuration: {best['Config']}")
    print(f"  AUC: {best['AUC_mean']:.4f}")

    return df_resultats

# Ajouter une cible
df_finance['defaut'] = np.random.choice([0, 1], n, p=[0.95, 0.05])

resultats_scoring = acp_pour_scoring(df_finance, variables_acp, 'defaut')
```

---

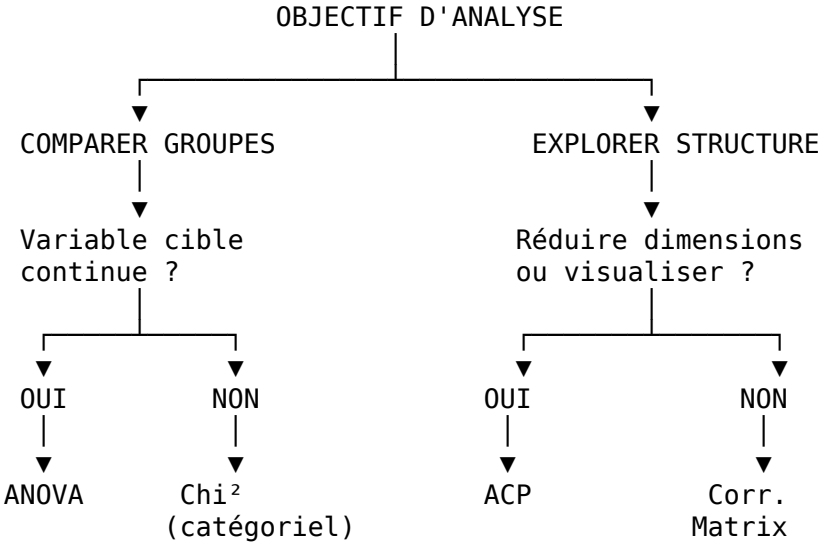
3. ANOVA vs ACP: Quand Utiliser Quoi?

3.1 Tableau Comparatif

Critère	ANOVA	ACP
Objectif	Comparer des groupes	Réduire les dimensions
Type d'analyse	Différences	Structure

Critère	ANOVA	ACP
Variable cible	OUI (continue)	NON
Variables explicatives	Catégorielles	Continues
Output	p-value, F-stat	Composantes, loadings
Question	“Y a-t-il une différence?”	“Quelle est la structure?”

3.2 Arbre de Décision



---

4. Résumé et Mnémotechniques

4.1 ANOVA

“FEP” pour l’ANOVA: - **F**-statistic: Plus grand = plus de différence - **E**ta-squared: Taille d’effet (0.01 petit, 0.06 moyen, 0.14 grand) - **P**ost-hoc: Si $p < 0.05$, faire Tukey

Conditions: “NIH” - **N**ormalité des distributions - **I**ndépendance des observations - **H**omogénéité des variances

4.2 ACP

“SCALE” pour l’ACP: - **S**tandardiser les données (OBLIGATOIRE) - **C**orrelation matrix à vérifier - **A**nalyser les loadings - **L**ooking at scree plot (coude) - **E**xpliquer les composantes

Critères de sélection: - **Kaiser**: $\lambda > 1$ - **Coude**: Point d’inflexion du scree plot - **80%**: Variance cumulée $\geq 80\%$

4.3 Erreurs à Éviter

Technique	Erreur courante	Solution
ANOVA	Ignorer les conditions	Tester avant (Levene, Shapiro)
ANOVA	Oublier post-hoc	Tukey si $p < 0.05$
ACP	Ne pas standardiser	Toujours StandardScaler
ACP	Interpréter les PC comme des variables	Les PC sont des combinaisons
Les deux	Confondre corrélation et causalité	Rester prudent