Test de Préparation: Cas Spéciaux - ACP, ANOVA, Outliers, Valeurs Manquantes

Informations

- **Durée estimée:** 50 minutes
- **Nombre de questions:** 30
- **Niveau:** Intermédiaire-Avancé
- **Thèmes:** ACP, ANOVA, Détection/Traitement outliers, Valeurs manquantes

---

Section 1: Valeurs Manquantes (8 questions)

Question 1

Quels sont les trois types de mécanismes de données manquantes?

A) Petit, Moyen, Grand
B) MCAR, MAR, MNAR
C) Avant, Pendant, Après
D) Simple, Double, Triple

Réponse

B) MCAR, MAR, MNAR

Type	Signification	Exemple
MCAR	Missing Completely At Random	Erreur de saisie aléatoire
MAR	Missing At Random	Revenu manquant selon l'âge
MNAR	Missing Not At Random	Riches ne déclarent pas leur revenu

Mnémotechnique: **"Les données MCARquent"** - MCAR: Le plus facile à traiter - MAR: Imputation conditionnelle possible - MNAR: Problématique, biais potentiel

---

Question 2

Quel est le risque principal de supprimer les lignes avec valeurs manquantes?

A) Le fichier devient plus léger
B) Perte d'information et biais potentiel si les données ne sont pas MCAR
C) Les calculs sont plus rapides
D) Aucun risque

Réponse

B) Perte d'information et biais potentiel si les données ne sont pas MCAR

Risques de la suppression (listwise deletion): - Réduction de la taille de l'échantillon - Perte de puissance statistique - Biais si les manquants ne sont pas aléatoires

```python
# Avant suppression
print(f"N avant: {len(df)}")

# Suppression
df_clean = df.dropna()

# Après
print(f"N après: {len(df_clean)}")
print(f"Perte: {(1 - len(df_clean)/len(df))*100:.1f}%")
```

Règle: Si > 5% de perte, considérer l'imputation.

Question 3

Quelle méthode d'imputation est la plus simple pour une variable numérique?

A) KNN Imputer
B) Imputation par la moyenne ou médiane
C) MICE
D) Régression

Réponse

B) Imputation par la moyenne ou médiane

```python
# Imputation simple
from sklearn.impute import SimpleImputer

# Par la moyenne
imputer_mean = SimpleImputer(strategy='mean')
df['revenu_imputed'] = imputer_mean.fit_transform(df[['revenu']])

# Par la médiane (plus robuste aux outliers)
imputer_median = SimpleImputer(strategy='median')
df['revenu_imputed'] = imputer_median.fit_transform(df[['revenu']])

# Par le mode (pour catégorielles)
imputer_mode = SimpleImputer(strategy='most_frequent')
df['segment_imputed'] = imputer_mode.fit_transform(df[['segment']])
```

⚠ Inconvénient: Réduit la variance de la variable.

Question 4

Qu'est-ce que l'imputation par groupe?

A) Imputer par la moyenne de toute la base
B) Imputer par la moyenne du groupe auquel appartient l'observation
C) Supprimer par groupe
D) Créer des groupes de manquants

Réponse

B) Imputer par la moyenne du groupe auquel appartient l'observation

```python
# Imputation par groupe (plus précise)
# Imputer le revenu par la médiane du segment
df['revenu_imputed'] = df.groupby('segment')['revenu'].transform(
    lambda x: x.fillna(x.median())
)

# Exemple:
# - Client Retail avec revenu manquant → médiane des Retail
# - Client Premium avec revenu manquant → médiane des Premium

# Combiné avec SimpleImputer
from sklearn.impute import SimpleImputer

for segment in df['segment'].unique():
    mask = df['segment'] == segment
    df.loc[mask, 'revenu'] = SimpleImputer(strategy='median').fit_transform(
        df.loc[mask, ['revenu']]
    )
```

C'est une forme de MAR: le manquant dépend du segment.

Question 5

Qu'est-ce que le KNN Imputer?

A) Un algorithme de classification
B) Imputation basée sur les K plus proches voisins
C) Suppression des K observations
D) Un test statistique

Réponse

B) Imputation basée sur les K plus proches voisins

```python
from sklearn.impute import KNNImputer

# KNN Imputer avec 5 voisins
imputer = KNNImputer(n_neighbors=5)
df_imputed = imputer.fit_transform(df[['age', 'revenu', 'anciennete']])

# Principe:
# 1. Trouver les K observations les plus similaires (sans considérer le manquant)
# 2. Imputer par la moyenne des valeurs de ces K voisins
```

Avantages: - Prend en compte les relations entre variables - Plus précis que la moyenne simple

Inconvénients: - Plus lent (calcul de distances) - Sensible aux outliers

---

Question 6

Qu'est-ce que MICE (Multiple Imputation by Chained Equations)?

A) Un animal
B) Une méthode d'imputation itérative multivariée
C) Un test de normalité

D) Une méthode de suppression

Réponse

B) Une méthode d'imputation itérative multivariée

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# MICE / IterativeImputer
mice_imputer = IterativeImputer(max_iter=10, random_state=42)
df_imputed = mice_imputer.fit_transform(df[['age', 'revenu', 'anciennete', 'score']])

# Principe:
# 1. Imputation initiale (moyenne)
# 2. Pour chaque variable avec manquants:
#    - Régression sur les autres variables
#    - Prédire les manquants
# 3. Répéter jusqu'à convergence
```

MICE est considéré comme l'état de l'art pour l'imputation complexe.

Question 7

Comment éviter le "data leakage" lors de l'imputation?

A) Imputer avant le split train/test
B) Fit l'imputer sur train, transform sur test
C) Imputer uniquement le test
D) Ne pas imputer

Réponse

B) Fit l'imputer sur train, transform sur test

```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Split AVANT imputation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Fit sur train uniquement
imputer = SimpleImputer(strategy='median')
imputer.fit(X_train)

# Transform sur les deux
X_train_imputed = imputer.transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Data leakage si:
# - Calcul de la moyenne sur tout le dataset avant split
# - Fit de l'imputer sur test
```

Le test doit simuler des données "futures" non vues.

Question 8

Une variable a 45% de valeurs manquantes. Quelle approche recommandez-vous?

A) Imputer par la moyenne
B) Supprimer la variable ou créer un indicateur de manquant
C) Ignorer le problème
D) KNN Imputer

Réponse

B) Supprimer la variable ou créer un indicateur de manquant

```python
# Analyse du taux de manquants
missing_pct = df.isnull().mean() * 100
print(missing_pct.sort_values(ascending=False))

# Décision selon le seuil
for col in df.columns:
    pct = df[col].isnull().mean() * 100
    if pct > 40:
        print(f"{col}: {pct:.1f}% → Considérer suppression ou indicateur")
    elif pct > 5:
        print(f"{col}: {pct:.1f}% → Imputer")
    else:
        print(f"{col}: {pct:.1f}% → Suppression des lignes acceptable")

# Créer un indicateur (le manquant peut être informatif!)
df['variable_manquante'] = df['variable'].isnull().astype(int)
```

À 45%, l'imputation risque d'introduire trop de bruit.

Section 2: Outliers (7 questions)

Question 9

Quelle est la règle IQR pour détecter les outliers?

A) Valeurs > moyenne + 2 × écart-type
B) Valeurs < Q1 - 1.5×IQR ou > Q3 + 1.5×IQR
C) Valeurs différentes de la médiane
D) Les 5% extrêmes

Réponse

B) Valeurs < Q1 - 1.5×IQR ou > Q3 + 1.5×IQR

```python
def detect_outliers_iqr(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1

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

    outliers = series[(series < lower_bound) | (series > upper_bound)]
    return outliers, lower_bound, upper_bound
```

```python
outliers, lb, ub = detect_outliers_iqr(df['montant'])
print(f"Bornes: [{lb:.2f}, {ub:.2f}]")
print(f"Outliers: {len(outliers)} ({len(outliers)/len(df)*100:.1f}%)")
```

Visualisation: Box plot montre directement les outliers.

Question 10

Quelle méthode de détection d'outliers est la plus robuste aux distributions non normales?

A) Z-score (> 3)
B) IQR
C) Moyenne ± 2σ
D) Minimum/Maximum

Réponse

B) IQR

Comparaison:

Méthode	Basée sur	Robuste?	Sensible aux outliers?
Z-score	Moyenne, σ	Non	Oui
IQR	Quartiles	Oui	Non
MAD	Médiane	Très	Non

```python
# MAD (Median Absolute Deviation) - encore plus robuste
from scipy.stats import median_abs_deviation

median = df['montant'].median()
mad = median_abs_deviation(df['montant'])

# Outliers: > 3 MAD de la médiane
threshold = 3
df['is_outlier_mad'] = (
    np.abs(df['montant'] - median) / mad > threshold
).astype(int)
```

Pour les données bancaires (revenus, montants), utiliser IQR ou MAD.

Question 11

Qu'est-ce que l'Isolation Forest?

A) Une forêt protégée
B) Un algorithme de détection d'anomalies basé sur l'isolement
C) Un type de tree plot
D) Une méthode de clustering

Réponse

B) Un algorithme de détection d'anomalies basé sur l'isolement

```python
from sklearn.ensemble import IsolationForest

# Isolation Forest
iso = IsolationForest(
    contamination=0.05,  # 5% d'outliers attendus
    random_state=42
)

# -1 = outlier, 1 = normal
df['anomaly'] = iso.fit_predict(df[['montant', 'nb_transactions', 'solde']])

# Filtrer les outliers
df_clean = df[df['anomaly'] == 1]
df_outliers = df[df['anomaly'] == -1]

print(f"Outliers détectés: {len(df_outliers)}")
```

Principe: Les anomalies sont plus faciles à isoler (moins de splits nécessaires).

Question 12

Qu'est-ce que le winsorizing (winsorisation)?

A) Supprimer les outliers
B) Remplacer les outliers par des valeurs seuils (caps)
C) Transformer en log
D) Calculer la moyenne

Réponse

B) Remplacer les outliers par des valeurs seuils (caps)

```python
from scipy.stats.mstats import winsorize

# Winsoriser à 5% de chaque côté
df['montant_winsor'] = winsorize(df['montant'], limits=[0.05, 0.05])

# Manuellement avec percentiles
p05 = df['montant'].quantile(0.05)
p95 = df['montant'].quantile(0.95)

df['montant_capped'] = df['montant'].clip(lower=p05, upper=p95)

# Résultat:
# - Valeurs < P05 remplacées par P05
# - Valeurs > P95 remplacées par P95
# - Valeurs entre P05 et P95 inchangées
```

Avantage: Conserve toutes les observations, réduit l'impact des extrêmes.

Question 13

Comment distinguer un outlier d'une observation valide mais rare?

A) Impossible à distinguer
B) Analyse contextuelle et expertise métier
C) Toujours supprimer
D) Toujours conserver

Réponse

B) Analyse contextuelle et expertise métier

```
# Exemple bancaire: Transaction de 500,000 HTG

# Statistiquement: Outlier (> Q3 + 1.5*IQR)

# Mais contextuellement:
# - Client VIP avec patrimoine de 50M HTG → Normal
# - Client Retail avec solde moyen de 10,000 HTG → Suspect (fraude?)

def evaluer_outlier(row, col, stats_segment):
    """Évaluer si l'outlier est contextuel"""
    segment = row['segment']
    valeur = row[col]

    # Comparer au segment, pas à la population globale
    p95_segment = stats_segment.loc[segment, 'p95']

    if valeur > p95_segment:
        if segment == 'VIP':
            return 'Outlier acceptable'
        else:
            return 'À investiguer'
    return 'Normal'

# Calculer les stats par segment
stats = df.groupby('segment')['montant'].quantile(0.95).rename('p95')
```

Question 14

Dans le contexte de la détection de fraude, comment traiter les outliers?

A) Les supprimer car ils faussent les analyses
B) Les conserver et les investiguer car ce sont potentiellement des fraudes
C) Les remplacer par la moyenne
D) Les ignorer

Réponse

B) Les conserver et les investiguer car ce sont potentiellement des fraudes

```
# En fraude, les outliers SONT l'objectif!

# Créer un flag pour investigation
df['flag_investigation'] = 0

# Règles métier
conditions = [
    (df['montant'] > df['montant_moyen_12m'] * 5),  # 5x la moyenne habituelle
```

```
    (df['nb_transactions_jour'] > 10),              # Plus de 10 trans/jour
    (df['pays'] != df['pays_residence']),           # Transaction à l'étranger
]

for condition in conditions:
    df.loc[condition, 'flag_investigation'] = 1

# Ces outliers sont envoyés à l'équipe anti-fraude
print(f"Transactions à investiguer: {df['flag_investigation'].sum()}")
```

En fraude: Outlier = Signal, pas Bruit!

Question 15

Quelle transformation peut réduire l'impact des outliers sur une régression?

A) One-Hot Encoding
B) Transformation logarithmique
C) Standardisation
D) Aucune

Réponse

B) Transformation logarithmique

```
import numpy as np

# Avant transformation
print(f"Max revenu: {df['revenu'].max():,.0f}")
print(f"Skewness: {df['revenu'].skew():.2f}")

# Transformation log
df['revenu_log'] = np.log1p(df['revenu'])  # log(1+x) pour gérer les 0

print(f"Max revenu_log: {df['revenu_log'].max():.2f}")
print(f"Skewness après: {df['revenu_log'].skew():.2f}")

# La transformation log:
# - Réduit l'écart entre petites et grandes valeurs
# - Rend la distribution plus symétrique
# - Diminue l'influence des outliers
```

Autres options: Racine carrée, Box-Cox, régression robuste.

Section 3: ANOVA (7 questions)

Question 16

Quand utiliser l'ANOVA?

A) Pour comparer deux moyennes
B) Pour comparer trois groupes ou plus
C) Pour tester la corrélation
D) Pour les variables catégorielles uniquement

Réponse

B) Pour comparer trois groupes ou plus

```
from scipy.stats import f_oneway

# ANOVA à un facteur: Comparer le revenu entre 3 segments
retail = df[df['segment'] == 'Retail']['revenu']
premium = df[df['segment'] == 'Premium']['revenu']
vip = df[df['segment'] == 'VIP']['revenu']

stat, pvalue = f_oneway(retail, premium, vip)
print(f"F-statistic: {stat:.2f}")
print(f"P-value: {pvalue:.4f}")

# Si p < 0.05: Au moins un groupe diffère significativement
```

Pour 2 groupes: t-test Pour 3+ groupes: ANOVA

Question 17

Quelles sont les hypothèses de l'ANOVA?

A) Normalité, indépendance, homogénéité des variances
B) Linéarité, normalité, stationnarité
C) Grande taille d'échantillon uniquement
D) Aucune hypothèse

Réponse

A) Normalité, indépendance, homogénéité des variances

```
from scipy.stats import shapiro, levene

# 1. Normalité (par groupe)
for segment in ['Retail', 'Premium', 'VIP']:
    data = df[df['segment'] == segment]['revenu']
    stat, p = shapiro(data[:5000])  # Shapiro limité à 5000
    print(f"{segment}: Shapiro p = {p:.4f}")

# 2. Homogénéité des variances (Levene)
stat, p = levene(retail, premium, vip)
print(f"Levene p = {p:.4f}")
# p > 0.05 → Variances homogènes □

# 3. Indépendance: Design de l'étude (non testable statistiquement)
```

Si hypothèses violées → Kruskal-Wallis (non-paramétrique)

Question 18

Le test de Levene donne p = 0.02. Que faire?

A) Procéder avec l'ANOVA classique
B) Utiliser l'ANOVA de Welch ou Kruskal-Wallis

C) Ignorer le résultat
D) Augmenter l'échantillon

Réponse

B) Utiliser l'ANOVA de Welch ou Kruskal-Wallis

p = 0.02 < 0.05 → Variances NON homogènes (hétéroscédasticité)

```
from scipy.stats import kruskal
import pingouin as pg

# Option 1: ANOVA de Welch (robuste à l'hétéroscédasticité)
welch_anova = pg.welch_anova(data=df, dv='revenu', between='segment')
print(welch_anova)

# Option 2: Kruskal-Wallis (non-paramétrique)
stat, p = kruskal(retail, premium, vip)
print(f"Kruskal-Wallis: H = {stat:.2f}, p = {p:.4f}")
```

ANOVA de Welch ne suppose pas l'égalité des variances.

Question 19

L'ANOVA donne F = 15.3, p = 0.0001. Que conclure?

A) Tous les groupes sont différents
B) Au moins un groupe diffère, mais on ne sait pas lequel
C) Aucune différence
D) Le test a échoué

Réponse

B) Au moins un groupe diffère, mais on ne sait pas lequel

L'ANOVA est un test "omnibus" - elle dit SI il y a une différence, pas OÙ.

```
from scipy.stats import tukey_hsd

# Tests post-hoc pour identifier les paires différentes
result = tukey_hsd(retail, premium, vip)
print(result)

# Ou avec statsmodels
from statsmodels.stats.multicomp import pairwise_tukeyhsd

tukey = pairwise_tukeyhsd(df['revenu'], df['segment'])
print(tukey.summary())

# Résultat typique:
# Retail vs Premium: p = 0.001 (différent)
# Retail vs VIP: p = 0.0001 (différent)
# Premium vs VIP: p = 0.15 (pas différent)
```

Question 20

Qu'est-ce que le test post-hoc de Tukey?

A) Un test préliminaire avant ANOVA
B) Des comparaisons par paires après ANOVA significative
C) Un test de normalité
D) Un test de variance

Réponse

B) Des comparaisons par paires après ANOVA significative

```python
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Tukey HSD (Honestly Significant Difference)
tukey = pairwise_tukeyhsd(
    endog=df['revenu'],      # Variable dépendante
    groups=df['segment'],     # Groupes
    alpha=0.05               # Niveau de significativité
)

print(tukey.summary())

# Output:
#   group1   group2   meandiff   p-adj    reject
# -----------------------------------------
#   Retail   Premium   25000      0.001    True
#   Retail   VIP       80000      0.0001   True
#   Premium  VIP       55000      0.003    True
```

Tukey contrôle le taux d'erreur familial (évite l'inflation du risque alpha).

Question 21

Comment interpréter l'eta-carré (η^2) en ANOVA?

A) C'est le p-value
B) C'est la proportion de variance expliquée par le facteur
C) C'est le nombre de groupes
D) C'est la moyenne des groupes

Réponse

B) C'est la proportion de variance expliquée par le facteur

```python
import pingouin as pg

# ANOVA avec taille d'effet
anova_result = pg.anova(data=df, dv='revenu', between='segment')
print(anova_result)

# η² (eta-squared) = SS_between / SS_total

# Interprétation:
# η² < 0.01: Effet négligeable
# η² = 0.01-0.06: Petit effet
```

```
# η² = 0.06-0.14: Effet moyen
# η² > 0.14: Grand effet

# Exemple: η² = 0.25
# → Le segment explique 25% de la variance du revenu
```

C'est l'équivalent du R^2 en régression.

Question 22

Qu'est-ce que l'ANOVA à deux facteurs?

A) ANOVA avec deux groupes
B) ANOVA avec deux variables indépendantes et leur interaction
C) Deux tests ANOVA séparés
D) ANOVA sur deux variables dépendantes

Réponse

B) ANOVA avec deux variables indépendantes et leur interaction

```python
import statsmodels.api as sm
from statsmodels.formula.api import ols

# ANOVA à deux facteurs: Segment × Genre
model = ols('revenu ~ C(segment) + C(genre) + C(segment):C(genre)', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)

# Interprétation:
# - Effet principal Segment: Différence entre Retail/Premium/VIP
# - Effet principal Genre: Différence entre M/F
# - Interaction Segment:Genre: L'effet du segment dépend-il du genre?

# Exemple d'interaction:
# - Hommes VIP gagnent 2x plus que Hommes Retail
# - Femmes VIP gagnent 1.5x plus que Femmes Retail
# → Interaction significative (l'effet VIP diffère selon le genre)
```

Section 4: ACP - Analyse en Composantes Principales (8 questions)

Question 23

Quel est l'objectif principal de l'ACP?

A) Prédire une variable cible
B) Réduire la dimensionnalité tout en conservant l'information
C) Classifier les observations
D) Tester des hypothèses

Réponse

B) Réduire la dimensionnalité tout en conservant l'information

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

# Données originales: 10 variables
X = df[['var1', 'var2', ..., 'var10']]

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

# ACP
pca = PCA(n_components=3)  # Réduire à 3 composantes
X_pca = pca.fit_transform(X_scaled)

# Variance expliquée
print(f"Variance expliquée: {pca.explained_variance_ratio_.sum()*100:.1f}%")
# Ex: 3 composantes capturent 85% de l'information de 10 variables
```

Question 24

Pourquoi faut-il standardiser les données avant l'ACP?

A) Pour accélérer le calcul
B) Pour que les variables avec grande variance ne dominent pas
C) Pour créer plus de composantes
D) Ce n'est pas nécessaire

Réponse

B) Pour que les variables avec grande variance ne dominent pas

```python
# Sans standardisation:
# - Revenu (en millions): variance = 1,000,000,000
# - Age (en années): variance = 150
# → Le revenu dominera totalement les composantes!

# Avec standardisation:
# - Toutes les variables ont variance = 1
# → Contribution équitable

from sklearn.preprocessing import StandardScaler

# TOUJOURS standardiser avant ACP
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Vérification
print(f"Moyennes après standardisation: {X_scaled.mean(axis=0)}")  # ~0
print(f"Écarts-types: {X_scaled.std(axis=0)}")  # ~1
```

Question 25

Comment choisir le nombre de composantes à retenir?

A) Toujours prendre 2
B) Critère de Kaiser (valeurs propres > 1) ou règle du coude
C) Prendre toutes les composantes
D) Choisir aléatoirement

Réponse

B) Critère de Kaiser (valeurs propres > 1) ou règle du coude

```python
# Méthode 1: Critère de Kaiser (eigenvalue > 1)
pca = PCA()
pca.fit(X_scaled)
eigenvalues = pca.explained_variance_
n_kaiser = (eigenvalues > 1).sum()
print(f"Composantes selon Kaiser: {n_kaiser}")

# Méthode 2: Règle du coude (scree plot)
import matplotlib.pyplot as plt
plt.plot(range(1, len(eigenvalues)+1), eigenvalues, 'bo-')
plt.axhline(y=1, color='r', linestyle='--', label='Kaiser threshold')
plt.xlabel('Composante')
plt.ylabel('Valeur propre')
plt.title('Scree Plot')
plt.legend()
plt.show()

# Méthode 3: Variance cumulative > 80%
cumsum = np.cumsum(pca.explained_variance_ratio_)
n_80 = (cumsum < 0.80).sum() + 1
print(f"Composantes pour 80% variance: {n_80}")
```

Question 26

Qu'est-ce que le "loading" en ACP?

A) Le temps de chargement
B) La corrélation entre une variable originale et une composante
C) Le nombre d'observations
D) La variance totale

Réponse

B) La corrélation entre une variable originale et une composante

```python
# Loadings = coefficients des variables dans chaque composante
loadings = pd.DataFrame(
    pca.components_.T,
    columns=[f'PC{i+1}' for i in range(pca.n_components_)],
    index=X.columns
)
print(loadings)

# Exemple:
#                PC1     PC2     PC3
# revenu        0.85    0.12   -0.05
# patrimoine    0.80    0.15    0.08
```

```
# age         0.10    0.75    0.20
# anciennete  0.05    0.78    0.15
# nb_produits 0.25    0.10    0.85

# Interprétation:
# PC1 = "Richesse" (revenu, patrimoine fortement corrélés)
# PC2 = "Maturité" (age, ancienneté)
# PC3 = "Engagement" (nb_produits)
```

Question 27

Le test de Bartlett en ACP teste quoi?

A) La normalité
B) Si les variables sont suffisamment corrélées pour justifier l'ACP
C) Le nombre de composantes
D) La variance expliquée

Réponse

B) Si les variables sont suffisamment corrélées pour justifier l'ACP

```python
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity

# Test de Bartlett (sphéricité)
chi_square, p_value = calculate_bartlett_sphericity(X)
print(f"Chi² = {chi_square:.2f}, p = {p_value:.4f}")

# H0: Matrice de corrélation = matrice identité (pas de corrélation)
# H1: Variables corrélées

# Si p < 0.05: Rejeter H0 → ACP appropriée
# Si p > 0.05: Variables non corrélées → ACP inutile
```

Si les variables ne sont pas corrélées, l'ACP ne peut pas les compresser.

Question 28

Qu'est-ce que le KMO (Kaiser-Meyer-Olkin)?

A) Un type de composante
B) Une mesure de l'adéquation des données pour l'ACP
C) La variance expliquée
D) Le nombre optimal de composantes

Réponse

B) Une mesure de l'adéquation des données pour l'ACP

```python
from factor_analyzer.factor_analyzer import calculate_kmo

# KMO
kmo_all, kmo_model = calculate_kmo(X)
print(f"KMO global: {kmo_model:.3f}")
```

```
# Interprétation du KMO:
# < 0.50: Inacceptable (ne pas faire d'ACP)
# 0.50-0.60: Misérable
# 0.60-0.70: Médiocre
# 0.70-0.80: Moyen
# 0.80-0.90: Bon
# > 0.90: Excellent

# Mnémotechnique: "KMO > 0.6 = OK pour ACP"
```

Question 29

Comment utiliser l'ACP avant un modèle de machine learning?

A) Remplacer les variables originales par les composantes
B) Ajouter les composantes aux variables originales
C) Utiliser uniquement PC1
D) L'ACP ne s'utilise pas avant ML

Réponse

A) Remplacer les variables originales par les composantes

```
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

# Pipeline: StandardScaler → PCA → Modèle
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=5)),  # Réduire à 5 composantes
    ('classifier', LogisticRegression())
])

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

# Avantages:
# - Réduction de la dimensionnalité
# - Élimination de la multicolinéarité
# - Potentiellement meilleure généralisation
```

Utile quand: - Beaucoup de variables corrélées - Risque d'overfitting - Besoin de visualisation (2-3 composantes)

Question 30

Comment interpréter un biplot en ACP?

A) C'est un nuage de points simple
B) Observations projetées + vecteurs des variables originales
C) Histogramme des composantes
D) Matrice de corrélation

Réponse

B) Observations projetées + vecteurs des variables originales

```python
import matplotlib.pyplot as plt

# Créer le biplot
fig, ax = plt.subplots(figsize=(10, 8))

# 1. Scatter des observations
ax.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.5)

# 2. Vecteurs des loadings
for i, var in enumerate(X.columns):
    ax.arrow(0, 0,
             pca.components_[0, i] * 3,
             pca.components_[1, i] * 3,
             head_width=0.1, color='red')
    ax.text(pca.components_[0, i] * 3.5,
            pca.components_[1, i] * 3.5,
            var, color='red')

ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_title('Biplot ACP')
plt.show()

# Interprétation:
# - Points proches = observations similaires
# - Vecteurs dans la même direction = variables corrélées
# - Vecteurs opposés = variables négativement corrélées
# - Longueur du vecteur = contribution à ces composantes
```

Résumé et Mnémotechniques

Valeurs Manquantes: “IMDS”

```
I - Identifier: % manquants, pattern
M - Mécanisme: MCAR, MAR, MNAR
D - Décider: Supprimer ou Imputer
S - Stratégie: Simple, Groupe, KNN, MICE
```

Outliers: “DITR”

```
D - Détecter: IQR, Z-score, Isolation Forest
I - Investiguer: Contexte métier
T - Traiter: Supprimer, Winsoriser, Transformer
R - Retester: Vérifier l'impact sur le modèle
```

ANOVA: “HAP”

```
H - Hypothèses: Normalité, Indépendance, Homogénéité
A - ANOVA: F-test omnibus
```

```
P - Post-hoc: Tukey si significatif
```

ACP: “SKRIP”

```
S - Standardiser: Obligatoire!
K - KMO & Bartlett: Vérifier l'adéquation
R - Réduire: Choisir le nombre de composantes
I - Interpréter: Loadings et variance expliquée
P - Projeter: Utiliser les composantes
```

Score et Auto-évaluation

Score	Niveau	Recommandation
27-30	Expert	Prêt pour l’examen
21-26	Avancé	Réviser les points faibles
15-20	Intermédiaire	Revoir les documents complets
< 15	Débutant	Étude approfondie nécessaire

Test préparé pour l’examen Data Analyst - UniBank Haiti Thème: Cas Spéciaux - Maîtriser les défis des données réelles