

Test Analyse Univariée et Multivariée - Test 1

Sujet: Analyses Univariées et Multivariées

Niveau: Intermédiaire

Nombre de questions: 25

Questions et Réponses

Q1. Quelle est la différence entre analyse univariée, bivariée et multivariée?

R1. | Type | Variables | Objectif | Exemple | ---|---|---|---| | **Univariée** | 1 | Décrire une seule variable | Distribution des montants | | **Bivariée** | 2 | Relation entre 2 variables | Score vs Défaut | | **Multivariée** | 3+ | Relations complexes | Scoring avec 10 prédicteurs |

Q2. Comment réaliser une analyse univariée complète d'une variable numérique?

R2.

```
def univariate_analysis(df, col):
    print(f"== Analyse de {col} ==")

    # Statistiques
    print(f"N: {df[col].count()}")
    print(f"Missing: {df[col].isnull().sum()}")
    print(f"Mean: {df[col].mean():.2f}")
    print(f"Median: {df[col].median():.2f}")
    print(f"Std: {df[col].std():.2f}")
    print(f"Skewness: {df[col].skew():.2f}")
    print(f"Kurtosis: {df[col].kurtosis():.2f}")

    # Visualisation
    fig, axes = plt.subplots(1, 3, figsize=(15, 4))

    # Histogramme
    axes[0].hist(df[col], bins=30, edgecolor='black')
    axes[0].set_title('Distribution')

    # Box plot
    axes[1].boxplot(df[col].dropna())
    axes[1].set_title('Box Plot')

    # QQ plot
    from scipy import stats
    stats.probplot(df[col].dropna(), plot=axes[2])
    axes[2].set_title('QQ Plot')

    plt.tight_layout()
    plt.show()

univariate_analysis(df, 'montant')
```

Q3. Comment analyser une variable catégorielle?

R3.

```
def analyze_categorical(df, col):
    # Fréquences
    freq = df[col].value_counts()
    pct = df[col].value_counts(normalize=True) * 100

    summary = pd.DataFrame({
        'Count': freq,
        'Percentage': pct.round(2)
    })
    print(summary)

    # Visualisation
    fig, ax = plt.subplots(figsize=(10, 5))
    df[col].value_counts().plot(kind='bar', ax=ax)
    ax.set_title(f'Distribution de {col}')
    plt.xticks(rotation=45)
    plt.show()

analyze_categorical(df, 'secteur')
```

Q4. Comment analyser la relation entre deux variables numériques?

R4.

```
def bivariate_numeric(df, x, y):
    # Corrélation
    pearson = df[x].corr(df[y], method='pearson')
    spearman = df[x].corr(df[y], method='spearman')

    print(f"Pearson: {pearson:.3f}")
    print(f"Spearman: {spearman:.3f}")

    # Scatter plot
    plt.figure(figsize=(10, 6))
    plt.scatter(df[x], df[y], alpha=0.5)

    # Ligne de tendance
    z = np.polyfit(df[x].dropna(), df[y].dropna(), 1)
    p = np.poly1d(z)
    plt.plot(df[x].sort_values(), p(df[x].sort_values()), "r--")

    plt.xlabel(x)
    plt.ylabel(y)
    plt.title(f'{y} vs {x} (r={pearson:.3f})')
    plt.show()

bivariate_numeric(df, 'score_credit', 'montant')
```

Q5. Comment analyser une variable numérique vs une variable catégorielle?

R5.

```
def bivariate_num_cat(df, num_var, cat_var):
    # Statistiques par groupe
    stats = df.groupby(cat_var)[num_var].describe()
    print(stats)

    # Box plots
    plt.figure(figsize=(12, 6))
    df.boxplot(column=num_var, by=cat_var)
    plt.title(f'{num_var} par {cat_var}')
    plt.suptitle('') # Remove auto title
    plt.show()

    # Test statistique (ANOVA)
    from scipy.stats import f_oneway
    groups = [group[num_var].dropna().values for name, group in df.groupby(cat_var)]
    f_stat, p_value = f_oneway(*groups)
    print(f"\nANOVA: F={f_stat:.2f}, p={p_value:.4f}")

bivariate_num_cat(df, 'montant', 'secteur')
```

Q6. Comment analyser la relation entre deux variables catégorielles?

R6.

```
def bivariate_categorical(df, cat1, cat2):
    # Tableau de contingence
    contingency = pd.crosstab(df[cat1], df[cat2], margins=True)
    print("Tableau de contingence:")
    print(contingency)

    # Pourcentages en ligne
    pct_row = pd.crosstab(df[cat1], df[cat2], normalize='index') * 100
    print("\nPourcentages en ligne:")
    print(pct_row.round(2))

    # Test Chi-carré
    from scipy.stats import chi2_contingency
    chi2, p, dof, expected = chi2_contingency(pd.crosstab(df[cat1], df[cat2]))
    print(f"\nChi-carré: ²={chi2:.2f}, p={p:.4f}, df={dof}")

    # Heatmap
    plt.figure(figsize=(10, 8))
    sns.heatmap(pd.crosstab(df[cat1], df[cat2]), annot=True, fmt='d', cmap='Blues')
    plt.title(f'{cat1} vs {cat2}')
    plt.show()

bivariate_categorical(df, 'secteur', 'defaut')
```

Q7. Qu'est-ce que l'ACP (Analyse en Composantes Principales) et à quoi sert-elle?

R7. L'ACP est une technique de réduction de dimensionnalité qui:

- Transforme les variables corrélées en composantes non-correlées
- Maximise la variance expliquée
- Réduit le nombre

de dimensions

Utilisations: - Visualisation de données haute dimension - Réduction de bruit - Traitement de la multicolinéarité - Préparation pour clustering/ML

Q8. Comment implémenter une ACP en Python?

R8.

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

# 1. Standardiser les données
features = ['var1', 'var2', 'var3', 'var4', 'var5']
X = df[features]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 2. Appliquer PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# 3. Variance expliquée
print("Variance expliquée par composante:")
print(pca.explained_variance_ratio_)
print(f"\nVariance cumulée: {pca.explained_variance_ratio_.cumsum()}")

# 4. Scree plot
plt.figure(figsize=(10, 5))
plt.bar(range(1, len(pca.explained_variance_ratio_)+1),
        pca.explained_variance_ratio_, label='Individuelle')
plt.plot(range(1, len(pca.explained_variance_ratio_)+1),
         pca.explained_variance_ratio_.cumsum(), 'r-o', label='Cumulée')
plt.xlabel('Composante')
plt.ylabel('Variance expliquée')
plt.legend()
plt.show()
```

Q9. Comment interpréter les résultats d'une ACP?

R9.

```
# Loadings (contributions des variables)
loadings = pd.DataFrame(
    pca.components_.T,
    columns=[f'PC{i+1}' for i in range(pca.n_components_)],
    index=features
)
print("Loadings:")
print(loadings.round(3))

# Interprétation:
# - Valeurs élevées (>0.5 ou <-0.5): variable contribue fortement
# - Même signe: variables corrélées positivement
```

```

# - Signes opposés: variables corrélées négativement

# Biplot
fig, ax = plt.subplots(figsize=(12, 8))
ax.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.5)
for i, var in enumerate(features):
    ax.arrow(0, 0, loadings.iloc[i, 0]*3, loadings.iloc[i, 1]*3,
             head_width=0.1, color='red')
    ax.text(loadings.iloc[i, 0]*3.5, loadings.iloc[i, 1]*3.5, var, color='red')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
plt.show()

```

Q10. Qu'est-ce que le clustering K-means?

R10. K-means est un algorithme de clustering qui: - Partitionne les données en K clusters - Minimise la variance intra-cluster - Assigne chaque point au centroïde le plus proche

Algorithme: 1. Initialiser K centroïdes aléatoirement 2. Assigner chaque point au centroïde le plus proche 3. Recalculer les centroïdes 4. Répéter 2-3 jusqu'à convergence

Q11. Comment implémenter K-means et choisir le nombre de clusters?

R11.

```

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

# Méthode du coude
inertias = []
silhouettes = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(X_scaled, kmeans.labels_))

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

# Coude
axes[0].plot(K_range, inertias, 'bx-')
axes[0].set_xlabel('K')
axes[0].set_ylabel('Inertia')
axes[0].set_title('Méthode du Coude')

# Silhouette
axes[1].plot(K_range, silhouettes, 'rx-')
axes[1].set_xlabel('K')
axes[1].set_ylabel('Silhouette Score')
axes[1].set_title('Score Silhouette')

```

```
plt.show()
```

Q12. Comment interpréter les clusters?

R12.

```
# Appliquer K-means avec K optimal
kmeans = KMeans(n_clusters=4, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled)

# Profil des clusters
cluster_profile = df.groupby('cluster').agg({
    'montant': 'mean',
    'anciennete': 'mean',
    'nb_produits': 'mean',
    'defaut': 'mean',
    'client_id': 'count'
}).rename(columns={'client_id': 'count'})

print("Profil des clusters:")
print(cluster_profile.round(2))

# Nommer les clusters selon leur profil
cluster_names = {
    0: 'Clients Premium',
    1: 'Nouveaux Clients',
    2: 'Clients à Risque',
    3: 'Clients Standard'
}
df['segment'] = df['cluster'].map(cluster_names)
```

Q13. Qu'est-ce que l'analyse RFM?

R13. **RFM** = Recency, Frequency, Monetary

Dimension	Définition	Mesure
Recency	Quand dernier achat/transaction	Jours depuis dernière tx
Frequency	Combien de transactions	Nombre de tx
Monetary	Combien dépensé	Montant total

Utilisation: Segmentation client pour marketing ciblé.

Q14. Comment implémenter une analyse RFM?

R14.

```
# Calcul RFM
reference_date = df['date_tx'].max() + pd.Timedelta(days=1)

rfm = df.groupby('client_id').agg({
```

```

'date_tx': lambda x: (reference_date - x.max()).days,
'tx_id': 'count',
'montant': 'sum'
})
rfm.columns = ['Recency', 'Frequency', 'Monetary']

# Scoring (1-5)
rfm['R_Score'] = pd.qcut(rfm['Recency'], 5, labels=[5,4,3,2,1])
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])

# Score combiné
rfm['RFM_Score'] = rfm['R_Score'].astype(str) + rfm['F_Score'].astype(str) + rfm['M_Score'].astype(str)

# Segmentation
def segment_rfm(row):
    if row['R_Score'] >= 4 and row['F_Score'] >= 4:
        return 'Champions'
    elif row['R_Score'] >= 3 and row['F_Score'] >= 3:
        return 'Loyaux'
    elif row['R_Score'] <= 2 and row['F_Score'] >= 3:
        return 'À Risque'
    else:
        return 'Standard'

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

```

Q15. Qu'est-ce que la régression multiple?

R15. Régression multiple: Modèle linéaire avec plusieurs prédicteurs.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n$$

Interprétation: - β_0 : Intercept (Y quand tous $X = 0$) - β_i : Changement de Y pour 1 unité de X_i , **toutes autres variables constantes**

Q16. Comment implémenter et interpréter une régression multiple?

R16.

```

import statsmodels.api as sm

# Préparer les données
X = df[['score_credit', 'revenu', 'anciennete']]
X = sm.add_constant(X) # Ajouter intercept
y = df['montant']

# Ajuster le modèle
model = sm.OLS(y, X).fit()
print(model.summary())

# Interprétation:
# - R²: Variance expliquée
# - Coef: Impact sur Y

```

```
# - p-value: Significativité  
# - IC: Intervalle de confiance
```

Q17. Comment vérifier les hypothèses d'une régression?

R17.

```
# 1. Linéarité: Résidus vs fitted  
residuals = model.resid  
fitted = model.fittedvalues  
plt.scatter(fitted, residuals)  
plt.axhline(y=0, color='r')  
plt.xlabel('Fitted')  
plt.ylabel('Residuals')  
  
# 2. Normalité des résidus  
from scipy import stats  
stats.probplot(residuals, plot=plt)  
  
# 3. Homoscédasticité: Résidus constants  
# (visuel: pas de pattern en entonnoir)  
  
# 4. Multicolinéarité: VIF  
from statsmodels.stats.outliers_influence import variance_inflation_factor  
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

Q18. Qu'est-ce que l'analyse discriminante linéaire (LDA)?

R18. LDA est une technique qui:
- Sépare les groupes en maximisant la variance inter-classes
- Réduit la dimensionnalité (comme PCA mais supervisée)
- Utilisée pour classification et visualisation

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis  
  
lda = LinearDiscriminantAnalysis()  
X_lda = lda.fit_transform(X_scaled, y)  
  
# Visualisation 2D  
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap='viridis')  
plt.xlabel('LD1')  
plt.ylabel('LD2')  
plt.show()
```

Q19. Comment réaliser une analyse de correspondance?

R19. L'analyse de correspondance est l'équivalent de l'ACP pour les variables catégorielles.

```
import prince  
  
# Analyse de correspondance simple  
ca = prince.CA(n_components=2)  
ca = ca.fit(contingency_table)  
  
# Coordonnées
```

```

row_coords = ca.row_coordinates(contingency_table)
col_coords = ca.column_coordinates(contingency_table)

# Biplot
ax = ca.plot_coordinates(
    X=contingency_table,
    ax=None,
    x_component=0,
    y_component=1
)

```

Q20. Comment évaluer la qualité d'un clustering?

R20. Métriques internes (sans labels):

```

from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score

# Silhouette (-1 à 1, plus haut = mieux)
sil = silhouette_score(X_scaled, labels)

# Calinski-Harabasz (plus haut = mieux)
ch = calinski_harabasz_score(X_scaled, labels)

# Davies-Bouldin (plus bas = mieux)
db = davies_bouldin_score(X_scaled, labels)

```

Métriques externes (avec vrais labels):

```

from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

ari = adjusted_rand_score(true_labels, predicted_labels)
nmi = normalized_mutual_info_score(true_labels, predicted_labels)

```

Q21. Comment identifier les variables les plus importantes dans un modèle?

R21.

```

# 1. Corrélation avec la cible
correlations = df.corrwith(df['target']).abs().sort_values(ascending=False)

# 2. Feature importance (Random Forest)
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)
importance = pd.DataFrame({
    'feature': X.columns,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)

# 3. Coefficients de régression (standardisés)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
# Ajuster régression sur données standardisées

```

Q22. Comment visualiser les données multidimensionnelles?

R22.

```
# 1. Pair plot
sns.pairplot(df[features + ['target']], hue='target')

# 2. PCA en 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')

# 3. t-SNE (non-linéaire)
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis')

# 4. Heatmap de corrélation
sns.heatmap(df[features].corr(), annot=True, cmap='RdBu_r')

# 5. Parallel coordinates
from pandas.plotting import parallel_coordinates
parallel_coordinates(df[features + ['cluster']], 'cluster')
```

Q23. Qu'est-ce que la régression logistique et quand l'utiliser?

R23. Régression logistique: Modèle de classification binaire.

$P(Y=1) = 1 / (1 + e^{-(z)})$
où $z = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n$

Utilisations bancaires: - Prédiction de défaut - Propensité à acheter un produit - Churn prediction

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict_proba(X_test)[:, 1]
```

Q24. Comment interpréter les odds ratios en régression logistique?

R24. Odds Ratio = e^{β_i}

```
import statsmodels.api as sm

X = sm.add_constant(df[['score', 'revenu']])
model = sm.Logit(df['defaut'], X).fit()

# Coefficients et Odds Ratios
results = pd.DataFrame({
    'coef': model.params,
    'odds_ratio': np.exp(model.params),
```

```

        'p_value': model.pvalues
    })
print(results)

```

Interprétation: - OR = 1: Pas d'effet - OR = 1.5: 50% de risque supplémentaire pour +1 unité - OR = 0.8: 20% de risque en moins pour +1 unité

Q25. Réalisez une analyse complète pour segmenter les clients d'une banque avec les variables: ancienneté, nombre de produits, solde moyen, nombre de transactions.

R25.

```

# 1. PRÉPARATION
features = ['anciennete', 'nb_produits', 'solde_moyen', 'nb_transactions']
X = df[features]

# 2. STANDARDISATION
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. ANALYSE UNIVARIÉE
for col in features:
    print(f"\n{col}:")
    print(df[col].describe())

# 4. MATRICE DE CORRÉLATION
print("\nCorrélations:")
print(df[features].corr().round(2))

# 5. PCA EXPLORATOIRE
pca = PCA()
pca.fit(X_scaled)
print(f"\nVariance expliquée: {pca.explained_variance_ratio_.cumsum()}")

# 6. CLUSTERING
# Méthode du coude
inertias = [KMeans(n_clusters=k, random_state=42).fit(X_scaled).inertia_
            for k in range(2, 8)]

# Appliquer K=4 (exemple)
kmeans = KMeans(n_clusters=4, random_state=42)
df['segment'] = kmeans.fit_predict(X_scaled)

# 7. PROFILS
profile = df.groupby('segment')[features].mean()
print("\nProfil des segments:")
print(profile.round(2))

# 8. NOMMER LES SEGMENTS
# Basé sur l'analyse des profils
segment_names = {
    0: 'Nouveaux (faible ancienneté, peu de produits)',
    1: 'Premium (haut solde, beaucoup de tx)',
    2: 'Dormants (haute ancienneté, peu de tx)',
}

```

```
    3: 'Actifs Standard (moyens partout)'  
}  
df['segment_name'] = df['segment'].map(segment_names)
```

Scoring

Score	Niveau
0-10	À améliorer
11-17	Intermédiaire
18-22	Avancé
23-25	Expert