TP – ACP (Analyse en Composantes Principales) – L2

Durée indicative : 1h30–2h
Objectif : appliquer pas à pas une ACP en Python, produire les graphiques clés (**scree plot**, **variance cumulée**, **biplot**, **cercle des corrélations**), utiliser le **critère de Kaiser**, et **interpréter**.

1) Jeu de données

Choisissez **l'une** des options :

- **Option A (sans internet, recommandé)** : *Wine* de scikit-learn (13 variables numériques).
- **Option B** : votre propre CSV (Kaggle, perso…) avec **≥ 6–10 colonnes numériques** (retirez les colonnes texte avant l'ACP).

2) Mise en place (imports)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

3) Charger les données

Option A – Wine (aucune connexion requise)

```
from sklearn.datasets import load_wine
wine = load_wine()
df = pd.DataFrame(wine.data, columns=wine.feature_names)
```

Option B – Votre CSV

```
# df = pd.read_csv("votre_fichier.csv")
# Garder uniquement les colonnes numériques :
# df = df.select_dtypes(include=[np.number])
```

4) Préparation des données

- Retirer/traiter les valeurs manquantes.
- **Standardiser** (centrer-réduire) avant l'ACP.

```
X = df.select_dtypes(include=[np.number]).copy()
X = X.dropna()  # simple : retire les lignes avec NA (sinon: imputer)

feature_names = X.columns.tolist()

scaler = StandardScaler()
Z = scaler.fit_transform(X)  # Z : données centrées-réduites (n × p)
```

À vérifier et commenter (2–3 lignes) :

- Nombre de lignes (n) et colonnes (p)
- Variables retenues (liste `feature_names`)
- S'il restait des NA

5) Lancer l'ACP

```
pca = PCA()                          # calcule toutes les composantes
scores = pca.fit_transform(Z)        # projections des individus (n × p)

eigvals = pca.explained_variance_          # ~ "valeurs propres"
ratio  = pca.explained_variance_ratio_     # % de variance par composante
cum    = ratio.cumsum()                     # variance expliquée cumulée
```

6) Courbe des éboulis (Scree plot)

But : repérer le **coude** → après ce point, chaque axe apporte peu d'info.

```
plt.figure()
plt.plot(range(1, len(ratio)+1), ratio*100, marker='o')
plt.xlabel("Composante principale"); plt.ylabel("Variance expliquée (%)")
plt.title("Courbe des éboulis (Scree plot)")
plt.xticks(range(1, len(ratio)+1))
plt.show()
```

À interpréter (3–5 lignes) : où est le **coude** ? combien d'axes semblent utiles ?

7) Variance expliquée cumulée

But : choisir le plus petit **K** qui atteint **70–90%** de variance cumulée.

```
plt.figure()
plt.plot(range(1, len(cum)+1), cum*100, marker='o')
plt.axhline(80, linestyle='--')  # exemple de seuil 80%
plt.xlabel("Nombre de composantes"); plt.ylabel("Variance expliquée cumulée
(%)")
plt.title("Variance expliquée cumulée")
plt.xticks(range(1, len(cum)+1))
plt.show()
```

À interpréter (2–3 lignes) : quel est le **plus petit K** pour atteindre 80% ?

8) Critère de Kaiser (sur corrélations ⇒ après standardisation)

Règle : garder les composantes avec $\lambda > 1$.

```
# Matrice de corrélation (variables standardisées)
R = np.corrcoef(Z, rowvar=False)

# Valeurs propres de R (R symétrique → eigh)
```

```
eigvals_R, eigvecs_R = np.linalg.eigh(R)
eigvals_R = eigvals_R[::-1]  # tri décroissant

print("Valeurs propres de R :", np.round(eigvals_R, 3))
print("Nb d'axes selon Kaiser (λ>1) :", (eigvals_R > 1.0).sum())
```

À interpréter (2–3 lignes) : combien d'axes selon **Kaiser** ? Comparez avec **coude** + **cumulée**.

9) Biplot (PC1 vs PC2)

Points = individus ; **flèches = variables** (importance + sens).

```
# Loadings = corr(variable, PC) pour données standardisées
loadings = pca.components_.T * np.sqrt(pca.explained_variance_)  # (p, p)

def biplot(scores2d, loads2d, names):
    plt.figure()
    plt.scatter(scores2d[:,0], scores2d[:,1], alpha=0.7)
    scale = 5.0  # agrandir les flèches pour les voir
    for i, name in enumerate(names):
        x, y = loads2d[i,0]*scale, loads2d[i,1]*scale
        plt.arrow(0, 0, x, y, head_width=0.15, length_includes_head=True)
        plt.text(x*1.05, y*1.05, name)
    plt.axhline(0); plt.axvline(0)
    plt.xlabel("PC1"); plt.ylabel("PC2"); plt.title("Biplot (PC1 vs PC2)")
    plt.show()

biplot(scores[:, :2], loadings[:, :2], feature_names)
```

À interpréter (5–8 lignes) :

- Quelles **variables** ont des **flèches longues** (bien expliquées par PC1–PC2) ?
- Quelles paires de variables semblent **corrélées** (flèches proches) ou **opposées** ?
- Y a-t-il des **groupes d'individus** visibles ? lesquels et pourquoi (au vu des flèches) ?

10) Cercle des corrélations (PC1 vs PC2)

Chaque flèche = corrélation de la variable avec **PC1** (x) et **PC2** (y).
Près du bord ⇒ très bien expliquée par PC1–PC2 ; **près de l'origine** ⇒ peu expliquée.

```
def correlation_circle(loads2d, names):
    plt.figure()
    ax = plt.gca()
    circle = plt.Circle((0,0), 1, fill=False)
    ax.add_artist(circle)
    for i, name in enumerate(names):
        x, y = loads2d[i,0], loads2d[i,1]
        plt.scatter(x, y)
        plt.arrow(0, 0, x, y, head_width=0.03, length_includes_head=True)
        plt.text(x*1.08, y*1.08, name)
    plt.axhline(0); plt.axvline(0)
    plt.xlim(-1.1, 1.1); plt.ylim(-1.1, 1.1); plt.gca().set_aspect("equal",
"box")
    plt.xlabel("PC1 (corr)"); plt.ylabel("PC2 (corr)"); plt.title("Cercle
des corrélations")
```

```
    plt.show()

correlation_circle(loadings[:, :2], feature_names)
```

À interpréter (5–8 lignes) :

- Quelles variables sont **près du bord** (fortement expliquées) ?
- Les **angles** entre flèches confirment-ils des corrélations (proches/opposées/90°) ?
- Quelles variables sont **mal expliquées** (près de 0,0) → faudra regarder **PC3/PC4** ?

11) Choisir K et refitter (optionnel, propre)

Fixez **K** (d'après coude + cumulée + Kaiser) puis refaites l'ACP avec **K** composantes.

```
K = 3  # exemple
pcaK = PCA(n_components=K).fit(Z)
scoresK = pcaK.transform(Z)
# Vous pouvez ensuite tracer les individus sur PC1-PC2, ou PC2-PC3, etc.
```