# TP0

## Bibliothèques utilisées

```
import pandas as pd      # pour la manipulation de données
import numpy as np       # pour les valeurs numériques, comme np.NaN
```

---

## Création de DataFrame

```
dF = pd.DataFrame(data, index= [...])
```

- pd.DataFrame() : créer une structure de tableau avec index personnalisés.
- np.NaN : représente une valeur manquante.

---

## Accès aux données

- dF.head(n) : affiche les n premières lignes.
- dF[['col1', 'col2']] : sélection de colonnes.
- dF.loc['index'] : accès par index *alphabétique*.
- dF.iloc[n] : accès par index *numérique*.
- dF.shape : retourne (nb_lignes, nb_colonnes).

---

## Filtrage des données

```
dF[dF['Attempts'] > 2]
dF[(dF['Score'] >= 15) & (dF['Score'] <= 20)]
dF[dF['Score'].isna()]        # lignes avec Score = NaN
```

---

## Modification des données

```
dF.loc['d', 'Score'] = 11.5       # modifier une valeur
dF['qualify'] = dF['qualify'].map({"yes": True, "no": False})  # conversion
dF['Name'] = dF['Name'].replace('James', 'Suresh')  # remplacer un nom
```

---

## Ajout ou suppression

```
df = pd.concat([dF, new_row_df])  # ajout d'une ligne
dF.drop(['k'])                    # suppression d’une ligne
dF = dF.drop(columns=['Attempts']) # suppression de colonne
```

---

## Statistiques

```
dF['Attempts'].sum()
dF['Score'].mean()
```

---

## Tri

```
dF.sort_values(by='Name')
dF.sort_values(by='Score', ascending=False)
```

---

## Parcours des lignes

```
for index, row in dF.iterrows():
   print(row['Name'], row['Score'])
```

---

## Remplissage & Nettoyage

```
dF.fillna(0, inplace=True)
dF.isna().sum()        # nombre de NaN par colonne
```

---

## Renommage et autres manipulations

```
dF.columns.tolist()                # liste des colonnes
dF.rename(columns={'Name': 'Full_name', 'Score': 'Number', 'qualify': '?'})
dF = dF[dF.columns[::-1]]          # inversion des colonnes
```

## 🎓 Résumé des fonctions importantes :

| Fonction | Utilité |
|---|---|
| `read_csv()` | Charger un fichier CSV |
| `describe()` | Statistiques descriptives |
| `isna().sum()` | Compter les valeurs manquantes |
| `drop_duplicates()` | Supprimer les doublons |
| `fillna(valeur)` | Remplacer les NaN |
| `map(dict)` | Encoder les valeurs catégorielles |
| `mean()` / `std()` | Moyenne / écart-type |
| `cov()` / `corr()` | Matrice de covariance / corrélation |
| `to_datetime(..., errors='coerce')` | Conversion intelligente en dates |

# TP1

## 📌 1. Importation et Chargement des Données

```
import pandas as pd
import numpy as np

dF = pd.read_csv('/empl.csv')     # Chargement de la base de données CSV
```

💡 **Astuce** : Utiliser .head() pour un aperçu rapide du DataFrame.

---

## 📂 2. Accès aux fichiers avec Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

➞ Permet d'accéder aux fichiers stockés sur Google Drive dans Google Colab.

---

## 📊 3. Statistiques descriptives

```
dF.describe()
```

➞ Fournit un résumé statistique des colonnes numériques (moyenne, écart type, min, max, etc.)

---

## 4. Détection des valeurs manquantes (NaN)

```
dF.isna().sum()
```

➞ Affiche le **nombre de valeurs manquantes** par colonne.

---

## 📅 5. Vérification et conversion des dates

```
dF['Date_emb'] = pd.to_datetime(dF['Date_emb'], errors='coerce')
```

➞ Transforme les chaînes en dates. Les erreurs deviennent NaT (Not a Time).

```
dF = dF[dF['Date_emb'].notna()]
```

➞ Supprime les lignes où Date_emb est NaT.

---

## 6. Gestion des doublons

```
dF.duplicated()
```

➞ Vérifie les doublons ligne par ligne.

```
dF.drop_duplicates()
```

➞ Supprime les doublons.

---

## 7. Remplissage des valeurs manquantes

```
dF.fillna(0)
```

➞ Remplace tous les NaN par 0.

---

## ⚠ 8. Correction d'une valeur aberrante

```
max_val = dF['Salaire'].max()
dF.loc[dF['Salaire'] == 0, 'Salaire'] = max_val
```

➞ Remplace les salaires nuls par le salaire **maximum** trouvé.

---

## 🔢 9. Encodage des valeurs catégorielles

```
Dict = {'manger': 0, 'Ingenieur': 1, 'Developpeur': 2}
dF['poste'] = dF['poste'].map(Dict)
```

➞ Transforme les postes en **valeurs numériques**.

---

## 📐 10. Centrage des données (matrice Y)

Y = dF[['Salaire','nb heures trav']] - dF[['Salaire','nb heures trav']].mean()

➞ On soustrait la moyenne de chaque colonne pour centrer les données.

---

## 📏 11. Centrage et réduction (matrice Z)

Z = (dF[['Salaire','nb heures trav']] - dF[['Salaire','nb heures trav']].mean()) / dF[['Salaire','nb heures trav']].std()

➞ Transformation des données en **score Z** (centrées et normalisées).

---

## 📈 12. Matrice de covariance

Z.cov()

➞ Montre les **covariances** entre les variables. Cela reflète la **variabilité conjointe**.

---

## 📉 13. Matrice de corrélation

Z.corr()

➞ Donne le **degré de corrélation linéaire** entre les colonnes : entre -1 (négatif) et +1 (positif).

# TP3

## 📚 1. Bibliothèques utilisées

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

---

## 📂 2. Chargement des données

```
from google.colab import drive
drive.mount('/content/drive')

X = pd.read_csv('/content/drive/MyDrive/cardiac.csv')
```

---

## 🔍 3. Nettoyage des données

- **Valeurs manquantes** :

```
X.isnull().sum()
```

- **Doublons** :

```
X.duplicated().sum()
```

- **Suppression de colonnes inutiles** :

```
X = X.drop(['Id'], axis=1, errors='ignore')
```

- **Remplacement des NaN par la moyenne** :

```
X = X.fillna(X.mean())
```

---

## ⚖️ 4. Standardisation

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- Vérification de la standardisation :

```
print(X_scaled.mean())  # ≈ 0
print(X_scaled.std())   # ≈ 1
```

---

## 🔗 5. Analyse de corrélation

- Création d'un DataFrame standardisé :

```
X1 = pd.DataFrame(X_scaled, columns=X.columns)
correlation = X1.corr()
```

- **Heatmap de la corrélation** :

```
sns.heatmap(correlation, annot=True, cmap='coolwarm')
```

- **Top 5 corrélations fortes (hors diagonale)** :

```
corr_pairs = correlation.unstack()
corr_pairs = corr_pairs[abs(corr_pairs) < 1].sort_values(ascending=False)
print(corr_pairs.head(5))
```

---

## 📉 6. ACP (Analyse en Composantes Principales)

- **Initialisation et transformation** :

```
pca = PCA()
Z_pca = pca.fit_transform(X)
```

- **Nombre de composantes principales** :

```
pca.n_components_
```

- **DataFrame des composantes principales** :

```
df_pca = pd.DataFrame(Z_pca, columns=[f'PC{i+1}' for i in range(pca.n_components_)])
```

- **Matrice de corrélation entre les nouvelles composantes** :

```
sns.heatmap(df_pca.corr(), annot=True, cmap='coolwarm')
```

---

## 📊 7. Analyse des valeurs propres

- **Valeurs propres** :

pca.explained_variance_

- **Scree plot (courbe de l'éboulis)** :

plt.plot(range(1, len(eigenvalues)+1), eigenvalues, marker='o')

- **Proportions de variances expliquées & cumulées** :

pca.explained_variance_ratio_
pca.explained_variance_ratio_.cumsum()

- **Variance cumulée ≥ 80%** :

n_components_80 = np.argmax(pca.explained_variance_ratio_.cumsum() >= 0.8) + 1

---

**□ 8. Projection des données**

- **Plan factoriel (PC1 vs PC2)** :

plt.scatter(df_pca['PC1'], df_pca['PC2'])

# TP4

## 1. Bibliothèques utilisées

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
```

---

## 2. Chargement & préparation des données

```
from google.colab import drive
drive.mount('/content/drive')

X = pd.read_csv('/content/drive/MyDrive/fromage.csv')
X = X.drop(['Fromages'], axis=1, errors='ignore')  # Variable qualitative supprimée
```

---

## 3. Standardisation des données

```
scaler = StandardScaler()
Z = scaler.fit_transform(X)  # Matrice centrée-réduite
```

---

## 4. ACP (Analyse en Composantes Principales)

```
pca = PCA(n_components=2)
Y = pca.fit_transform(Z)  # Projection dans le plan PC1-PC2
```

---

## 5. K-Means Clustering (k = 3)

```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(Y)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

inertia = kmeans.inertia_

- **Étiquettes et répartition** :

pd.Series(labels).value_counts()

- **Visualisation** :

sns.scatterplot(x=Y[:,0], y=Y[:,1], hue=labels)
plt.scatter(centroids[:,0], centroids[:,1], c='black', s=200, marker='X')

---

## 📏 6. Distances aux centroïdes

distances = kmeans.transform(Y)  # Matrice des distances

---

## 🔍 7. Choix optimal de k

- **Méthode du coude & silhouette** :

```
inertias = []
silhouettes = []
for k in range(2, 7):
   kmeans = KMeans(n_clusters=k, random_state=42)
   kmeans.fit(Y)
   inertias.append(kmeans.inertia_)
   silhouettes.append(silhouette_score(Y, kmeans.labels_))
```

- **Tracé** :

```
plt.plot(K_range, inertias)          # Coude
plt.plot(K_range, silhouettes)       # Silhouette
```

- **k optimal selon silhouette** :

optimal_k = K_range[np.argmax(silhouettes)]

---

## 🌲 8. Classification Ascendante Hiérarchique (CAH)

linkage_matrix = linkage(Z, method='ward')

dendrogram(linkage_matrix, labels=X.index.tolist(), color_threshold=7)

```
plt.axhline(y=7, color='r', linestyle='--')
```

- **Découpage en groupes** :

```
groupes = fcluster(linkage_matrix, t=7, criterion='distance')
```

---

## 🔄 9. Comparaison CAH vs KMeans

```
X_groupes = pd.DataFrame({'Observation': X.index, 'Groupe': groupes})
X_groupes['KMeans'] = labels
pd.crosstab(X_groupes['Groupe'], X_groupes['KMeans'])  # Tableau croisé
```