Maaloul Mohamed Khalil

# Processus ECD 101

## Phases de prétraitement et de transformation

**lien fichier IPYNB:** "ECD.ipynb"

[1]:
```
import pandas as pd
```

1) Télécharger dans votre notebook la base de données ‘empl.csv

’[ ]:
```
#travail avec google colab
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
path = '/content/drive/empl.csv'
df = pd.read_csv(path)
```

[3]:
```
#avec jupyter local
df = pd.read_csv("empl.csv")
df
```

[3]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 0 | Alice | 30-30-3030 | manger | NaN | 18 |
| 1 | Bob | 15-02-2020 | Ingenieur | 90000.0 | 42 |
| 2 | Charlie | 01-06-2024 | Developpeur | NaN | 30 |
| 3 | David | 12-08-2020 | Ingenieur | 65000.0 | 38 |
| 4 | Emma | 16-05-2022 | Ingenieur | 83000.0 | 35 |
| 5 | Alice | 06-12-2023 | manger | 8000.0 | 18 |
| 6 | Bob | 15-02-2020 | Ingenieur | 90000.0 | 42 |
| 7 | marwa | 01-12-2013 | Developpeur | 73000.0 | 48 |

2) Interpréter le jeu de données en donnant le nombre des NaN et en décrivant les statistiques de base

[4]:
```
print("Les informations du dataframe :")
print(df.info())
print(f"Les statistiques de base du dataframe :")
print(df.describe())
print(f"Le nombre de valeurs NaN dans chaque colonne :")
print(df.isna().sum())
```

```
Les informations du dataframe :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Nom             8 non-null      object
 1   Date_emb        8 non-null      object
 2   poste           8 non-null      object
 3   Salaire         6 non-null      float64
 4   nb heures trav  8 non-null      int64
dtypes: float64(1), int64(1), object(3)
memory usage: 452.0+ bytes
None
Les statistiques de base du dataframe :
            Salaire  nb heures trav
count      6.000000         8.00000
mean   68166.666667        33.87500
std    31070.350282        11.14114
min     8000.000000        18.00000
25%    67000.000000        27.00000
50%    78000.000000        36.50000
75%    88250.000000        42.00000
max    90000.000000        48.00000
Le nombre de valeurs NaN dans chaque colonne :
Nom               0
Date_emb          0
poste             0
Salaire           2
nb heures trav    0
dtype: int64
```

3) Vérifier l'intégrité du domaine des valeurs de la variable 'Date_emb' pd.to_datetime(df['date'], errors='coerce') : convertit le format de la date en datetime et si une valeur n'est pas une date valide
(par exemple, un texte aléatoire ou un format incorrect),
elle est remplacée par NaT (Not a Time) .notna() → Élimine les valeurs NaT.

[5]:
```
df['Date_emb'] = pd.to_datetime(df['Date_emb'], errors='coerce')
df.head(3)
```

[5]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 0 | Alice | NaT | manger | NaN | 18 |
| 1 | Bob | 2020-02-15 | Ingenieur | 90000.0 | 42 |
| 2 | Charlie | 2024-01-06 | Developpeur | NaN | 30 |

[6]:
```
#verification
invalid_date = df[ df["Date_emb"].isna() ]
print("les dates invalides:")
print(invalid_date)
valid_date = df[ df["Date_emb"].notna() ]
print("\nles dates valides")
valid_date.head(3)
```

```
les dates invalides:
     Nom Date_emb   poste  Salaire  nb heures trav
0  Alice      NaT  manger      NaN              18

les dates valides
```

[6]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 1 | Bob | 2020-02-15 | Ingenieur | 90000.0 | 42 |
| 2 | Charlie | 2024-01-06 | Developpeur | NaN | 30 |
| 3 | David | 2020-12-08 | Ingenieur | 65000.0 | 38 |

4) Vérifier s'il existe des doublons dans votre base, si c'est le cas procédé à leur suppression .duplicated() / .drop_duplicates()

[7]:
```
doublons = valid_date[ valid_date.duplicated() ]
doublons
```

[7]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 6 | Bob | 2020-02-15 | Ingenieur | 90000.0 | 42 |

[8]:
```
df_sans_doublons = valid_date.drop_duplicates()
df_sans_doublons.tail(3)
```

[8]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 4 | Emma | 2022-05-16 | Ingenieur | 83000.0 | 35 |
| 5 | Alice | 2023-06-12 | manger | 8000.0 | 18 |
| 7 | marwa | 2013-01-12 | Developpeur | 73000.0 | 48 |

5) Remplacer toutes les valeurs NaN par Zéro

[9]:
```
df_sans_doublons = df_sans_doublons.fillna(0)
df_sans_doublons.head(3) #salaire 0 = valeur aberrantes
```

[9]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 1 | Bob | 2020-02-15 | Ingenieur | 90000.0 | 42 |
| 2 | Charlie | 2024-01-06 | Developpeur | 0.0 | 30 |
| 3 | David | 2020-12-08 | Ingenieur | 65000.0 | 38 |

6) Déduire si le traitement de la question précédente a conduit à l'apparition de valeurs aberrantes dans la base de données, si c'est le cas remplacer ces valeurs par la valeur max de la variable.

[10]:
```
df = df_sans_doublons
# Utilisation de .loc[] pour remplacer les valeurs de la colonne␣
 ↪"Salaire" qui sont égales à 0
# df.loc[condition, colonne] permet de sélectionner et de modifier␣
 ↪les lignes où une condition est vraie.
# Ici, on sélectionne les lignes où "Salaire" == 0 et on remplace␣
 ↪ces valeurs par max "Salaire".
df.loc[df["Salaire"] == 0, "Salaire"] = df["Salaire"].max()
df.head(3)
```

[10]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 1 | Bob | 2020-02-15 | Ingenieur | 90000.0 | 42 |
| 2 | Charlie | 2024-01-06 | Developpeur | 90000.0 | 30 |
| 3 | David | 2020-12-08 | Ingenieur | 65000.0 | 38 |

7) Appliquer le codage nécessaire pour transformer les valeurs de la variable Poste en des valeurs numériques. (Exemple ; manger :0, Ingénieur :1, Développeur : 2)

[11]:
```
df["poste"] = df["poste"].map({'manger': 0, 'Ingenieur': 1,␣
 ↪'Developpeur': 2})
df.head(3)
```

[11]:

| | Nom | Date_emb | poste | Salaire | nb heures trav |
|---|---|---|---|---|---|
| 1 | Bob | 2020-02-15 | 1 | 90000.0 | 42 |
| 2 | Charlie | 2024-01-06 | 2 | 90000.0 | 30 |
| 3 | David | 2020-12-08 | 1 | 65000.0 | 38 |

8) Calculer la matrice Y des données centrées et la matrice Z de données centrées et réduites

[12]:
```
X = df[["Salaire","nb heures trav"]]
# donnée centré Y = X - X bar
Y = X - X.mean()
print("\nMoyenne de Y (doit être proche de 0) :\n", Y.mean())
# donnée centré réduit Z = (X - X bar)/ecarttype
Z = (X - X.mean()) / X.std()
print("\nÉcart-type de Z (doit être proche de 1) :\n", Z.std())
```

```
Moyenne de Y (doit être proche de 0) :
 Salaire          -4.850638e-12
nb heures trav    2.368476e-15
dtype: float64

Écart-type de Z (doit être proche de 1) :
 Salaire           1.0
nb heures trav    1.0
dtype: float64
```

9) Calculer la matrice RX des corrélations de notre matrice de données

[13]:
```
R_X = X.corr()
print("\nMatrice R_X (corrélations) :\n", R_X)
```

```
Matrice R_X (corrélations) :
                  Salaire  nb heures trav
Salaire          1.000000         0.693091
nb heures trav   0.693091         1.000000
```

[14]:
```
print(f"la correlation entre ses deux variables
{round(R_X.iloc[0, 1], 2)} est proche de 1")
```

```
la correlation entre ses deux variables 0.69 est proche de 1
```

=> il existe forte correlation linéaire entre ses deux variables

et la matrice VZ des variances et covariances de Z. Commenter

[15]:
```
V_Z = Z.cov()
print("\nMatrice V_Z de covariances de Z :\n", V_Z)
```

```
Matrice V_Z de covariances de Z :
                  Salaire  nb heures trav
Salaire          1.000000         0.693091
nb heures trav   0.693091         1.000000
```

=> R_X et V_Z sont égales