2CSSID-TP01. Prétraitement

- Binôme 01 : ....
- Binôme 02 : ....

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from typing import Tuple
```

I. Réalisation des algorithmes

Cette partie sert à améliorer la compréhension des algorithmes de préparation de données vus en cours en les implémentant à partir de zéro. Pour ce faire, on va utiliser la bibliothèque numpy qui est utile dans les calcules surtout matricielles.

I.1. Normalisation

Ici, on va réaliser les deux fonctions de nomalisation : standard et min-max. On va prendre une matrice $X[N, M]$ de N échantillons et M colonnes. La normalisation standard d'une colonne j peut être décrite comme :

$$standard(X_j) = \frac{X_j - \mu(X_j)}{\sigma(X_j)}$$

La nomalisation min-max d'une colonne j peut être décrite comme :

$$minmax(X_j) = \frac{X_j - min(X_j)}{max(X_j) - min(X_j)}$$

In [3]:
```python
# TODO compléter la standardisation d'une matrice
# Entrée : la matrice des données (N échantillons X  M caractéristiques)
# Sortie : vecteur de M moyennes, vecteur de M écart-types, une matrice normalisée
def norm_std(X: np.ndarray) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    return None, None, None


#=========================================================================
# TEST UNITAIRE
#=========================================================================
# (array([4. , 3. , 0.5]),
#  array([1.87082869, 2.        , 0.5       ]),
#  array([[ 1.60356745,  1.        , -1.        ],
#         [-1.06904497, -1.        ,  1.        ],
#         [-0.53452248,  1.        , -1.        ],
#         [ 0.        , -1.        ,  1.        ]]))
#-------------------------------------------------------------------------

X = np.array([
    [7, 5, 0],
    [2, 1, 1],
    [3, 5, 0],
    [4, 1, 1],
])

norm_std(X)
```

Out[3]:
```
(array([4. , 3. , 0.5]),
 array([1.87082869, 2.        , 0.5       ]),
 array([[ 1.60356745,  1.        , -1.        ],
        [-1.06904497, -1.        ,  1.        ],
        [-0.53452248,  1.        , -1.        ],
        [ 0.        , -1.        ,  1.        ]]))
```

In [4]:
```python
# TODO compléter la standardisation d'une matrice
# Entrée : la matrice des données (N échantillons X  M caractéristiques)
# Sortie : vecteur de M max, vecteur de M min, une matrice normalisée
def norm_minmax(X: np.ndarray) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    return None, None, None
```

```
#========================================================================
# TEST UNITAIRE
#========================================================================
# (array([7, 5, 1]),
#  array([2, 1, 0]),
#  array([[1. , 1. , 0. ],
#         [0. , 0. , 1. ],
#         [0.2, 1. , 0. ],
#         [0.4, 0. , 1. ]]))
#------------------------------------------------------------------------

X = np.array([
    [7, 5, 0],
    [2, 1, 1],
    [3, 5, 0],
    [4, 1, 1],
])

norm_minmax(X)
```

Out[4]:
```
(array([7, 5, 1]),
 array([2, 1, 0]),
 array([[1. , 1. , 0. ],
        [0. , 0. , 1. ],
        [0.2, 1. , 0. ],
        [0.4, 0. , 1. ]]))
```

I.2. Encodage One-Hot

Etant donné un vecteur $A[N]$ représentant une caractéristique nominale donnée, on veut encoder les valeurs en utilisant One-Hot. Pour faciliter la tâche, on vous donne l'algorithme détaillé :

1. Trouver les valeurs uniques dans le vecteur A ; on appele ça : un vocabulaire V
2. Créer une matrice $X[N, |V|]enrecopiantlevecteurVN fois. Dans python, on peut recopier un vecteur en utilisant l'instruction : [V] * N
3. Comparer l'égalité entre chaque ligne de A et chaque ligne (qui est un vecteur) de X.
4. Transformer les booléens vers des entiers

In [5]:
```python
# TODO compléter l'encodage One-Hot
# Entrée : un vecteur d'une caractéristique (N échantillons)
# Sortie : vecteur du vocabulaire V, matrice N X |V|
def one_hot(A: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
    V = np.unique(A)
    return V, None

#=========================================================================
# TEST UNITAIRE
#=========================================================================
# (array(['COLD', 'HOT', 'MILD'], dtype='<U4'),
#  array([[0, 1, 0],
#         [0, 0, 1],
#         [1, 0, 0],
#         [0, 1, 0],
#         [0, 0, 1]]))
#-------------------------------------------------------------------------

A = np.array(['HOT', 'MILD', 'COLD', 'HOT', 'MILD'])
one_hot(A)
```

Out[5]:
```
(array(['COLD', 'HOT', 'MILD'], dtype='<U4'),
 array([[0, 1, 0],
        [0, 0, 1],
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]]))
```

I.3. Binarisation

Etant donné un vecteur $A[N]$ représentant une caractéristique numérique donnée, on veut encoder les valeurs en 0 ou 1 selon un seuil s. La binarization d'un élément A_i est donnée par :

$$A'_i = \begin{cases} 1 & \text{si } A_i \geq s \\ 0 & \text{sinon} \end{cases}$$

In [6]:
```python
# TODO compléter la binarisation
# Entrée : un vecteur d'une caractéristique (N échantillons), un nombre
```

```
# Sortie : un vecteur binarisé (N échantillons)
def bin(A: np.ndarray, seuil: float) -> np.ndarray:
    return None

#===============================================================================
# TEST UNITAIRE
#===============================================================================
# array([1, 0, 0, 0, 1, 1])
#-------------------------------------------------------------------------------

A = np.array([5, 2, 1, -1, 6, 4])

bin(A, 4)
```

Out[6]:
```
array([1, 0, 0, 0, 1, 1])
```

II. Application et analyse

Cette partie sert à appliquer les algorithmes, modifier les paramètres et analyser les résultats.

II.1. Lecture des données

On va lire 4 fichiers :

- un fichier CSV avec des colonnes séparées par des virgules
- un fichier CSV avec des colonnes séparées par des point-virgules
- un fichier Sqlite
- un fichier XML

In [7]:
```
adult1 = pd.read_csv("data/adult1.csv", skipinitialspace=True)
adult1.head(10)
```

Out[7]:

	age	workclass	education	Marital-status	occupation	sex	Hours-per-week	class
0	39.0	State-gov	Bachelors	Never-married	Adm-clerical	Male	40	<=50K
1	50.0	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Male	13	<=50K
2	38.0	Private	HS-grad	Divorced	Handlers-cleaners	Male	40	<=50K
3	53.0	Private	11th	NaN	Handlers-cleaners	Male	40	<=50K
4	28.0	Private	Bachelors	Married-civ-spouse	Prof-specialty	Female	40	<=50K
5	37.0	Private	Masters	Married-civ-spouse	Exec-managerial	Female	40	<=50K
6	49.0	Private	9th	Married-spouse-absent	Other-service	Female	16	<=50K
7	52.0	Self-emp-not-inc	HS-grad	Married-civ-spouse	Exec-managerial	Male	45	>50K
8	31.0	Private	Masters	Never-married	Prof-specialty	Female	50	>50K
9	42.0	Private	Bachelors	Married-civ-spouse	Exec-managerial	Male	40	>50K

In [8]:
```
noms = ["class", "age", "sex", "workclass", "education", "hours-per-week", "marital-status"]
adult2 = pd.read_csv("data/adult2.csv", skipinitialspace=True, sep=";", header=None, names=noms)
adult2.head(10)
```

Out[8]:

	class	age	sex	workclass	education	hours-per-week	marital-status
0	N	25	F	Private	Some-college	40	Married-civ-spouse
1	N	18	F	Private	HS-grad	30	Never-married
2	Y	47	F	Private, Prof-school	60	Married-civ-spouse	NaN
3	Y	50	M	Federal-gov	Bachelors	55	Divorced
4	N	47	M	Self-emp-inc	HS-grad	60	Divorced
5	Y	43	M	Private	Some-college	40	Married-civ-spouse
6	N	46	M	Private	5th-6th	40	Married-civ-spouse
7	N	35	M	Private	Assoc-voc	40	Married-civ-spouse
8	N	41	M	Private	HS-grad	48	Married-civ-spouse
9	N,30	M	Private, HS-grad	40	Married-civ-spouse	NaN	NaN

In [9]:

```
import sqlite3
#établir la connexion avec la base de données
con = sqlite3.connect("data/adult3.db")
#récupérer le résultat d'une réquête SQL sur cette connexion
adult3 = pd.read_sql_query("SELECT * FROM income", con)

#remplacer les valeurs "?" par NaN de numpy
adult3 = adult3.replace('?', np.nan)

adult3.head(10)
```

Out[9]:

	num	age	workclass	education	marital-status	sex	hours-per-day	class
0	1	76	Private	Masters	married	M	8.0	Y
1	2	44	Private	Bachelors	married	M	12.0	Y
2	3	47	Self-emp-not-inc	Masters	single	F	10.0	N
3	4	20	Private	Some-college	single	F	8.0	N
4	5	29	Private	HS-grad	single	M	8.0	N
5	6	32	Self-emp-inc	HS-grad	married	M	8.0	Y
6	7	17	NaN	10th	single	F	6.4	N
7	8	30	Private	11th	single	M	8.0	N
8	9	31	Local-gov	HS-grad	single	F	8.0	N
9	10	42	Private	HS-grad	married	M	8.0	N

In [10]:
```
!pip install lxml
```

```
Requirement already satisfied: lxml in /opt/penv/ml/lib/python3.8/site-packages (4.9.1)
```

In [11]:
```
from lxml import etree
#créer le parser et spécifier qu'il doit valider le DTD
parser = etree.XMLParser(dtd_validation=True)
#analyser le fichier XML en utilisant ce parser
arbre = etree.parse("data/adult4.xml", parser)

def valeur_noeud(noeud):
    return noeud.text if noeud is not None else np.nan

noms2 = ["id", "age", "workclass", "education", "marital-status", "sex", "hours-per-week", "class"]
adult4 = pd.DataFrame(columns=noms2)

for candidat in arbre.getroot():
    idi = candidat.get("id")
    age = valeur_noeud(candidat.find("age"))
```

```
    workclass = valeur_noeud(candidat.find("workclass"))
    education = valeur_noeud(candidat.find("education"))
    marital = valeur_noeud(candidat.find("marital-status"))
    sex = valeur_noeud(candidat.find("sex"))
    hours = valeur_noeud(candidat.find("hours-per-week"))
    klass = valeur_noeud(candidat.find("class"))

    adult4 = pd.concat(
        [adult4,
         pd.Series([idi, age, workclass, education, marital, sex, hours, klass],index=noms2).to_frame().T
        ], axis=0, ignore_index=True)
adult4.head(10)
```

Out[11]:

	id	age	workclass	education	marital-status	sex	hours-per-week	class
0	52	47	Local-gov	Some-college	divorced	F	38	N
1	53	34	Private	HS-grad	single	F	40	N
2	54	33	Private	Bachelors	single	F	40	N
3	55	21	Private	HS-grad	single	M	35	N
4	56	52	NaN	HS-grad	divorced	M	45	Y
5	57	48	Private	HS-grad	married	M	46	N
6	58	23	Private	Bachelors	single	M	40	N
7	59	71	Self-emp-not-inc	Some-college	divorced	M	2	N
8	60	29	Private	HS-grad	divorced	M	60	N
9	61	42	Private	Bachelors	divorced	M	50	N

TODO: Analyse

- Que remarquez-vous concernant l'ordre, le nombre et les noms des caractéristiques dans les 4 datasets ?
- Que remarquez-vous à propos des valeurs dans les 4 tables ?

Réponse

- ...
- ...

II.2. Intégration des données

Dans cette section, on va appliquer des opérations sur les différentes tables. Vous devez à chaque fois figurer ce qu'on a fait et pourquoi.

In [12]:
```
# Afficher les noms des colonnes de adult3
list(adult3.columns)
```

Out[12]:
```
['num',
 'age',
 'workclass',
 'education',
 'marital-status',
 'sex',
 'hours-per-day',
 'class']
```

In [13]:
```
adult3.rename(columns={"num": "id", "hours-per-day": "hours-per-week"}, inplace=True)
adult1.rename(columns={"Hours-per-week": "hours-per-week", "Marital-status": "marital-status"}, inplace=True)

# Afficher les noms des colonnes de adult3
list(adult3.columns)
```

Out[13]:
```
['id',
 'age',
 'workclass',
 'education',
 'marital-status',
 'sex',
 'hours-per-week',
 'class']
```

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)
- Est-ce qu'en appliquant cette opération, on aura certains problèmes ?

Réponse

- ...
- ...
- ...

In [14]:
```
ordre = ["age", "workclass", "education", "marital-status", "sex", "hours-per-week", "class"]
adult1 = adult1.reindex(ordre + ["occupation"], axis=1)
#print adult1.head()
adult2 = adult2.reindex(ordre, axis=1)
adult3 = adult3.reindex(ordre + ["id"], axis=1)
adult4 = adult4.reindex(ordre + ["id"], axis=1)

# Afficher les noms des colonnes de adult3
list(adult3.columns)
```

Out[14]:
```
['age',
 'workclass',
 'education',
 'marital-status',
 'sex',
 'hours-per-week',
 'class',
 'id']
```

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...

- ...

In [15]:
```
# Afficher les deux premières lignes de la table adult3
adult3.head(2)
```

Out[15]:

	age	workclass	education	marital-status	sex	hours-per-week	class	id
0	76	Private	Masters	married	M	8.0	Y	1
1	44	Private	Bachelors	married	M	12.0	Y	2

In [16]:
```
adult3["hours-per-week"] *= 5

# Afficher les deux premières lignes de la table adult3
adult3.head(2)
```

Out[16]:

	age	workclass	education	marital-status	sex	hours-per-week	class	id
0	76	Private	Masters	married	M	40.0	Y	1
1	44	Private	Bachelors	married	M	60.0	Y	2

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [17]:
```
adult34 = pd.concat([adult3, adult4], ignore_index=True)
adult34.head(10)
```

Out[17]:

	age	workclass	education	marital-status	sex	hours-per-week	class	id
0	76	Private	Masters	married	M	40.0	Y	1
1	44	Private	Bachelors	married	M	60.0	Y	2
2	47	Self-emp-not-inc	Masters	single	F	50.0	N	3
3	20	Private	Some-college	single	F	40.0	N	4
4	29	Private	HS-grad	single	M	40.0	N	5
5	32	Self-emp-inc	HS-grad	married	M	40.0	Y	6
6	17	NaN	10th	single	F	32.0	N	7
7	30	Private	11th	single	M	40.0	N	8
8	31	Local-gov	HS-grad	single	F	40.0	N	9
9	42	Private	HS-grad	married	M	40.0	N	10

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [18]:
```
# Transformer le champs "id" à un entier
adult34["id"] = pd.to_numeric(adult34["id"], downcast="integer")
# Ordonner la table en se basant sur les valeurs de "id"
adult34 = adult34.sort_values(by="id")

# L'opération que vous devez deviner (une opération de vérification)
```

```
red = adult34[adult34.duplicated("id", keep=False)]
red
```

Out[18]:

	age	workclass	education	marital-status	sex	hours-per-week	class	id
44	70	Private	Some-college	single	M	40.0	N	45
94	70	Private	Some-college	single	M	8	N	45
45	31	Private	HS-grad	single	F	30.0	N	46
95	31	Private	HS-grad	single	NaN	6	N	46
46	22	Private	Some-college	married	M	24.0	N	47
96	22	Private	Some-college	married	M	4.8	N	47
47	36	Private	HS-grad	widowed	F	24.0	N	48
97	NaN	Private	HS-grad	widowed	F	4.8	N	48
48	64	Private	11th	married	M	40.0	N	49
98	64	Private	11th	married	M	8	N	49
49	43	NaN	Some-college	divorced	F	40.0	N	50
99	43	Federal-gov	Some-college	divorced	F	8	N	50

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [19]:
```
# Il y a un problème avec cette forme
# en attendant qu'il soit réglé
#adult34 = adult34.groupby("id").ffill()

adult34.update(adult34.groupby(['id']).ffill())
adult34.update(adult34.groupby(['id']).bfill())

# L'opération de vérification précédente
red = adult34[adult34.duplicated("id", keep=False)]
red
```

Out[19]:

	age	workclass	education	marital-status	sex	hours-per-week	class	id
44	70	Private	Some-college	single	M	40.0	N	45
94	70	Private	Some-college	single	M	8	N	45
45	31	Private	HS-grad	single	F	30.0	N	46
95	31	Private	HS-grad	single	F	6	N	46
46	22	Private	Some-college	married	M	24.0	N	47
96	22	Private	Some-college	married	M	4.8	N	47
47	36	Private	HS-grad	widowed	F	24.0	N	48
97	36	Private	HS-grad	widowed	F	4.8	N	48
48	64	Private	11th	married	M	40.0	N	49
98	64	Private	11th	married	M	8	N	49
49	43	Federal-gov	Some-college	divorced	F	40.0	N	50
99	43	Federal-gov	Some-college	divorced	F	8	N	50

TODO: Analyse

- Quelle opération a-t-on appliqué ?

- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [20]:
```
adult34.drop_duplicates("id", keep="last", inplace=True)

# On refait la même opération précédente
red = adult34[adult34.duplicated("id", keep=False)]
red
```

Out[20]:

	age	workclass	education	marital-status	sex	hours-per-week	class	id

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [21]:
```
list(adult1.columns)
```

Out[21]:
```
['age',
 'workclass',
 'education',
 'marital-status',
 'sex',
 'hours-per-week',
 'class',
 'occupation']
```

In [22]:
```
adult1.drop(["occupation"], axis=1, inplace=True)
adult34.drop(["id"], axis=1, inplace=True)

list(adult1.columns)
```

Out[22]:
```
['age',
 'workclass',
 'education',
 'marital-status',
 'sex',
 'hours-per-week',
 'class']
```

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [23]:
```
# les différentes valeurs du colonne adult1.marital-status
adult1["marital-status"].unique()
```

Out[23]:
```
array(['Never-married', 'Married-civ-spouse', 'Divorced', nan,
       'Married-spouse-absent', 'Separated', 'Married-AF-spouse'],
      dtype=object)
```

In [24]:
```
dic = {
    "Never-married": "single",
    "Married-civ-spouse": "married",
    "Married-spouse-absent": "married",
    "Married-AF-spouse": "married",
    "Divorced": "divorced",
    "Separated": "divorced",
    "Widowed": "widowed"
}
```

```
adult1["marital-status"] = adult1["marital-status"].map(dic)
adult2["marital-status"] = adult2["marital-status"].map(dic)

# les différentes valeurs du colonne adult1.marital-status après mappage
adult1["marital-status"].unique()
```

Out[24]:
```
array(['single', 'married', 'divorced', nan], dtype=object)
```

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [25]:
```
# On va appliquer la même opération sur d'autres caractéristiques
adult1["sex"] = adult1["sex"].map({"Female": "F", "Male": "M"})
adult1["class"] = adult1["class"].map({"<=50K": "N", ">50K": "Y"})

# Ensuite, on fusionne les tables dans une seule
adult = pd.concat([adult1, adult2, adult34], ignore_index=True)

# dimension de la table adult
adult.shape
```

Out[25]:
```
(194, 7)
```

II.3. Nétoyage des données

Ici, on va appliquer des opérations de nétoyage. C'est à vous de déviner quelle opération a-t-on utilisé et pourqoi.

In [26]:
```
# Afficher le nombre des valeurs nulles dans chaque colonne
adult.isnull().sum()
```

Out[26]:
```
age               5
workclass        10
education         1
marital-status    4
sex               2
hours-per-week    2
class             0
dtype: int64
```

In [27]:
```
adult.dropna(subset=["workclass", "education", "marital-status", "sex", "hours-per-week", "class"], inplace=True)
adult.isnull().sum()
```

Out[27]:
```
age               3
workclass         0
education         0
marital-status    0
sex               0
hours-per-week    0
class             0
dtype: int64
```

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

In [28]:
```
adult["age"] = pd.to_numeric(adult["age"])
adult["age"] = adult.groupby(["class", "education"])["age"].transform(lambda x: x.fillna(int(round(x.mean()))))
adult.isnull().sum()
```

Out[28]:
```
age               0
workclass         0
education         0
marital-status    0
sex               0
hours-per-week    0
class             0
dtype: int64
```

TODO: Analyse

- Quelle opération a-t-on appliqué ?
- Pourquoi ? (Quel est l'intérêt ?)

Réponse

- ...
- ...

II.4. Transformation des données

In [29]:
```
adult["education"].head(6)
```

Out[29]:
```
0    Bachelors
1    Bachelors
2      HS-grad
4    Bachelors
5      Masters
6          9th
Name: education, dtype: object
```

In [30]:
```
from sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
# le résultat c'est un numpy.ndarray
education_enc = ord_enc.fit_transform(adult[["education"]])
education_enc[:6,]
```

Out[30]:
```
array([[6.],
       [6.],
       [8.],
       [6.],
       [9.],
       [3.]])
```

TODO: Analyse

- Quel est le type d'encodage utilisé ?
- A votre avis, dans quel cas peut-on utiliser ce type d'encodage ?

Réponse

- ...
- ...

In [31]:
```
adult["sex"].head(6)
```

Out[31]:
```
0    M
1    M
2    M
4    F
5    F
6    F
Name: sex, dtype: object
```

In [32]:
```
from sklearn.preprocessing import OneHotEncoder
onehot_enc = OneHotEncoder()
# le résultat c'est un numpy.ndarray
sex_enc = onehot_enc.fit_transform(adult[["sex"]])
sex_enc.toarray()[:6,]
```

Out[32]:
```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

TODO: Analyse

- Quel est le type d'encodage utilisé ?
- A votre avis, dans quel cas peut-on utiliser ce type d'encodage ?

Réponse

- ...
- ...

In [33]:
```
adult["hours-per-week"] = pd.to_numeric(adult["hours-per-week"])
adult["hours-per-week"].head(3)
```

Out[33]:
```
0    40.0
1    13.0
2    40.0
Name: hours-per-week, dtype: float64
```

In [34]:
```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler()
# le résultat c'est un numpy.ndarray
hours_per_week_prop = min_max_scaler.fit_transform(adult[["hours-per-week"]])
hours_per_week_prop[:3,]
```

Out[34]:
```
array([[0.49367089],
       [0.15189873],
       [0.49367089]])
```

In [35]:
```
# pour ajouter la nouvelle caractéristique au dataframe
adult["hours-per-week-prop"] = hours_per_week_prop
adult.head(3)
```

Out[35]:

	age	workclass	education	marital-status	sex	hours-per-week	class	hours-per-week-prop
0	39.0	State-gov	Bachelors	single	M	40.0	N	0.493671
1	50.0	Self-emp-not-inc	Bachelors	married	M	13.0	N	0.151899
2	38.0	Private	HS-grad	divorced	M	40.0	N	0.493671

TODO: Analyse

- Comment la normalisation MinMax est calculée ?
- Décrire les valeurs résultats (plage de valeurs, etc.) ?
- Est-ce que les valeurs du dataset de test sont garanties d'être dans la plage ?
- Si oui, expliquer pouruoi. Si non, comment garantir la plage des valeurs ?

Réponse

- ...
- ...
- ...
- ...

In [36]:
```
adult["age"].head(3)
```

Out[36]:
```
0    39.0
1    50.0
2    38.0
Name: age, dtype: float64
```

In [37]:
```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
# le résultat c'est un numpy.ndarray
age_normal = min_max_scaler.fit_transform(adult[["age"]])
age_normal[:3,]
```

Out[37]:
```
array([[0.3442623 ],
       [0.52459016],
       [0.32786885]])
```

TODO: Analyse

- Comment la normalisation standard est calculée ?
- Décrire les valeurs résultats (plage de valeurs, etc.) ?

Réponse

- ...
- ...

In [38]:
```
adult["age"].head(10)
```

Out[38]:
```
0     39.0
1     50.0
2     38.0
4     28.0
5     37.0
6     49.0
7     52.0
8     31.0
9     42.0
10    43.0
Name: age, dtype: float64
```

In [39]:
```
from sklearn.preprocessing import Binarizer

binarizer = Binarizer(threshold=40)
# le résultat c'est un numpy.ndarray
age_bin = binarizer.fit_transform(adult[["age"]])
age_bin[:10,]
```

Out[39]:
```
array([[0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.]])
```

TODO: Analyse

- Quelle est l'opération appliquée ici ?
- Quel est son rôle ?

Réponse

- ...
- ...