# ***TP_2 _DATAMINING : Encodage***

## *Présentation des données

Nos analyses se feront sur la base « Iris », une base de données implémentée dans Python. Mains néanmoins dans notre travail, nous l'avons importé sous le format « csv » vers Python. Nous avons eu à importer le fichier « Iris.csv » et à le charger dans notre logiciel de travail « Python ».

```
#%% *) Chargement de la base de donnée iris.csv
import pandas as pd
data = pd.read_csv('C:/Users/DELL/Desktop/pytho_AS3/Iris .csv')
```

Pour se faire nous avons eu à importer la librairie « Panda » qui nous permettra de lire notre base « iris ». Ensuite nous avons importé notre base « Iris » que nous avons enregistrer dans l'objet « data ». Notre base comporte 150 lignes et 6 variables.

**Variables « Target »**

Notre travail consistera ici à encoder la variable dépendante de notre base. Parmi les 6 variables de notre base, nous allons nous intéresser seulement à la variable « species » qui contient les types de fleurs. Nous avons donc extrait cette variable de notre base dans la variable 'flower' et avons visualisé cette dernier à travers la deuxième ligne de notre code.

- Entrée

```
flower = data['Species']  # endogène  (target)
print(flower.value_counts())
```

- Sortie

```
In [213]: print(flower.value_counts())
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

On peut voir que notre variable dépendante est constituée de trois (3) modalités : Iris-sétosa, Iris-versicolor, Iris-virginica. On peut voir dans la sortie que chaque modalité comporte chacun 50 observations.

## Les types d'encodages : Théories et Pratique sur python et Avantages et limites

Ici nous allons nous intéresser aux travaux proprement dit : L'encodage des variables.
Nous axerons cette partie sur la présentation des différents encodages de python, la pratique sur python sans oublier de faire ressortir les avantages et inconvénients de ces encodages.

- LabelEncoder

Elle permet tous simplement de transformer les données qualitatives en numériques et ne fais intervenir aucun ordre dans l'encodage.
Elle est limitée du fait qu'elle ne peut transformer une grande base de données en des données encodées. Elle ne transforme que les colonnes pris individuellement ou spécifiés.

- Entrée

  Pour ce code on utilise le module « LabelEncoder que l'on importe de sklearn ».
  Ensuite une fois ce module recueilli on l'applique à notre base ou donnée transformée avec la fonction (fit_transform)

```
#%% I)Encodage des variable nominales comme notre cas
ECOD = LabelEncoder()
flow_trans = ECOD.fit_transform(flower)
print(flow_trans)
```

- Sortie

```
    ...: flow_trans = ECOD.fit_transform(flower)
    ...: print(flow_trans)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Une fois notre code exécuté on obtient ce dessus. Elle à procédé a un remplacement de nos modalités soit par 0, 1,2 qu'il s'agisse de sétosa, versicolor ou virginica

❖ *le One hot encoding

Le *one hot encoding* est la méthode la plus populaire pour transformer une variable catégorique en variable numérique. Sa popularité réside principalement dans la facilité d'application. De plus, pour beaucoup de problèmes, elle donne de bons résultats. Son principe est le suivant :
Considérons une variable catégorique X qui admet *K* modalités *m1*, *m2*, ..., *mK*. Le *one hot encoding* consiste à créer *K* variables indicatrices, soit un vecteur de taille *K* qui a des 0 partout et un 1 à la position *i* correspondant à la modalité *mi*. On remplace donc la variable catégorique par *K* variables numériques.

❖ *Exécution et visualisations sous python

- Entrée

Ici nous avons importé le module « OneHotEncoder » de sklearn. Ce qui est un peu frappant dans notre code est en premier lieu l'ajout de 'reshape(-1,1).toarray' dans notre code. En effet reshape nous permet de redimensionner nos données et de les transformer en tableau avec (toarray)
On a utilisé (-1,1) comme paramètre car l'on ne connait pas les dimensions de notre tableau donc ce paramètre attribuera à notre tableau la dimension qui lui ait adapté. La ligne suivante nous permettra de recueillir nos données recodées dans des colonnes différentes et associées a notre code

```
#%%
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
OHE = encoder.fit_transform(data.Species.values.reshape(-1,1)).toarray()
df_OH = pd.DataFrame(OHE, columns = ["Species_" + str(encoder.categories_[0][i])
                                     for i in range(len(encoder.categories_[0]))])
df_OH_final = pd.concat([data, df_OH], axis=1)
df_OH_final
```

- Sortie

| Species | Job Iris-setosa | Job Iris-versicolor | Job Iris-virginica |
|---|---|---|---|
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |
| Iris-virginica | 0 | 0 | 1 |

Pour la sortie, comme on peut le voir notre variable Species a été transformé en trois colonnes avec pour nom de colonnes les différentes modalités concernées. Donc au moment où la variable concernée est présente dans la variable d'origine elle mentionne 1 dans la colonne de cette modalité concernée et ainsi de suite pour les autres variables.

*__Avantages__ : simple, intuitif, rapide à mettre en place.

__Inconvénients__ : Lorsque le nombre de modalités est élevé (supérieur à 100 par exemple), le nombre de nouvelles variables créées est également élevé. Ainsi, on se retrouve avec un jeu de données beaucoup plus volumineux, qui occupe plus d'espace en mémoire et dont le traitement par les algorithmes d'apprentissage devient plus difficile. Aussi, certains algorithmes, notamment quelques implémentations des forêts d'arbres décisionnels, n'arrivent pas à exploiter au mieux les informations contenues dans ces variables lorsque ce nombre de modalités est trop grand.

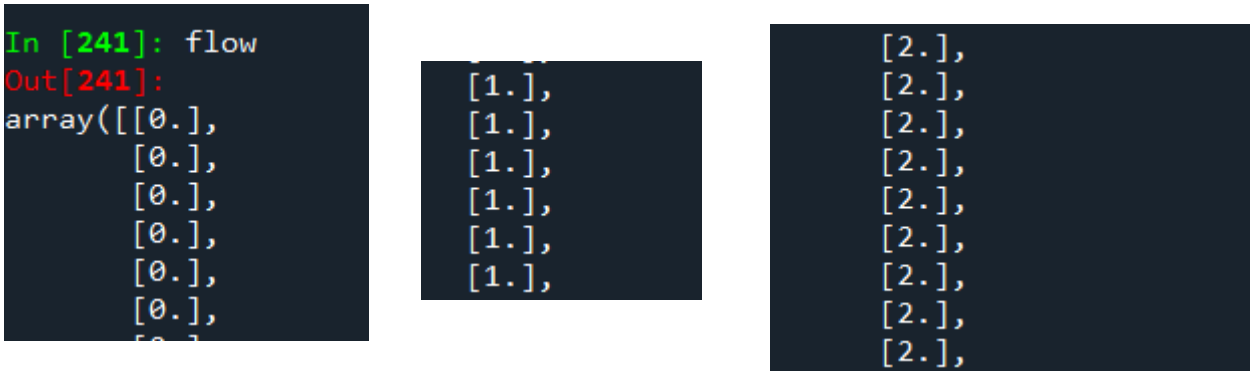
- *Ordinal Encoder

Cette dernière marche comme label Encoder mais seulement qu'elle tient en compte l'ordre des modalités et s'applique à toutes les données de la base. Pour des données de grandes dimensions il serait difficile d'appliquer cette dernière ou dans les  données catégorielles

- Entrée

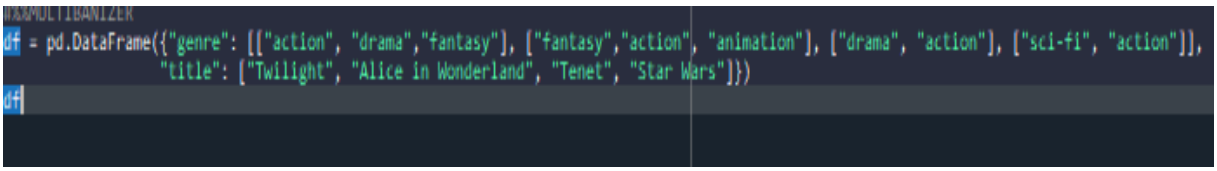
```
#%%Encodage des variables ordinales
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
flow = encoder.fit_transform(data.Species.values.reshape(-1,1))
flow
```

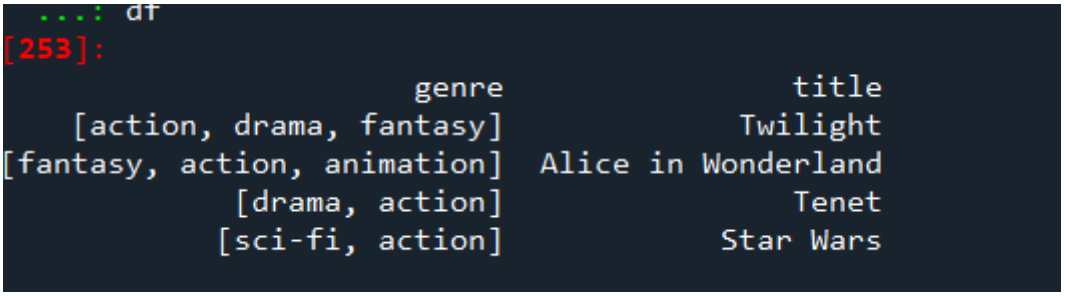
- Sorties




- **MultiLabelBinarizer**

C'est un cas particulier des encodages. Elle permet d'encoder des données de plusieurs modalités. Elle fonctionne comme le One Hot Encoder mais concerne les données a differnts modalités affectés. Notre base n'étant pas adapté à ce dernier encodage, nous aller créer des donnes et l'appliquer (confère code suivant)




- Exécution python

Elle utilise même principe que Hot Encoder.

- Entrée
  Ainsi le code s'exécute de la même manière que le Hot Encoder.

```
df
#%%MULTIBANIZER
from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer()

res = pd.DataFrame(mlb.fit_transform(df['genre']),
                   columns=mlb.classes_,
                   index=df['genre'].index)
res
```

- Sortie

On obtient des données encodées et séparés dans les colonnes

```
Out[256]:
   action  animation  drama  fantasy  sci-fi
0       1          0      1        1       0
1       1          1      0        1       0
2       1          0      1        0       0
3       1          0      0        0       1
```

*Rôle des encodages

On a cité plein de rôles dans les parties précédentes mais il faut retenir que le plus important de ses rôles est de faciliter l'analyse des données , les modélisations