



Pour aller plus loin...

Nettoyage & Préparation des données

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- Changer le nom d'une colonne
- Eliminer un caractère indésirable
- Obtenir plusieurs informations à partir des dates
(jour, mois, année, jour de la semaine, etc.)
- Transformation des données texte en donnée numérique
avec des encodeurs
- Regrouper des données en catégorie (« chaud », « froid »,
« cher », « abordable », « pas cher »)
- Normalisation des données





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Changer le nom d'une colonne

- **Difficulté à résoudre :**

- Parfois, les noms des colonnes qu'on retrouve dans les données ne sont pas très représentatifs (ou comportent même des erreurs)

- **Solution possible :**

- On peut changer les noms des colonnes d'un DataFrame grâce à l'opération « **rename** »

DataFrame à modifier

Dictionnaire avec colonnes à changer
{ ancien nom : nouveau nom }

inplace=True
Modifie le DataFrame

```
df.rename ( columns={ ' MONTANT ' : 'MONTANT' } , inplace=True )
```

```
df.rename ( columns= { df.columns[4] : 'montant' } , inplace=True )
```

On peut aussi se servir de l'attribut **columns** pour indiquer la colonne à supprimer par sa position
(première colonne = position [0])

Changer le nom d'une colonne



- Exemple : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - Sur le notebook « **ExoExtra1.ipynb** », on va lire le fichier « **VentesAgenceU.csv** »

```
import pandas as pd Ne pas oublier l'import
dfFactures = pd.read_csv
('http://kirschpm.fr/cours/PythonDataScience/files/VentesAgenceU.csv',
 delimiter=';', header=[0], index_col=[0]) Lecture du fichier
```

```
dfFactures.info()
```

info

Index: 50 entries, FA-2008-0010 to nan

Data columns (total 6 columns):

#	Column	Non-Null Count	Dtype
0	DATE FACTURE	47 non-null	object
1	CODE CLIENT	47 non-null	object
2	SECTEUR	47 non-null	object
3	VENDEUR	47 non-null	object
4	MONTANT	47 non-null	object
5	Unnamed: 6	0 non-null	float64

dtypes: float64(1), object(5)
 memory usage: 2.7+ KB

```
for col in dfFactures.columns :
    print('##{}##'.format(col))
```

Boucle pour afficher le nom de chaque colonne entre « ## » (et mieux voir les espaces)

```
##DATE FACTURE##
##CODE CLIENT##
##SECTEUR##
##VENDEUR##
## MONTANT ##
##Unnamed: 6##
```

Espace dans le nom de la colonne

Changer le nom d'une colonne



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - On modifie le nom de la colonne

```
dfFactures.rename ( columns={' MONTANT ':'MONTANT'}, inplace=True)
```

Ancien nom

Nouveau nom

On modifie le DataFrame

```
for col in dfFactures.columns :
    print('##{}##'.format(col))
```

```
dfFactures.info()
```

```
##DATE FACTURE##
```

```
##CODE CLIENT##
```

```
##SECTEUR##
```

```
##VENDEUR##
```

```
##MONTANT##
```

```
##Unnamed: 6##
```

Boucle pour afficher le nom de chaque colonne entre « ## » (et mieux voir les espaces)

Nouveau nom de la colonne

info

#	Column	Non-Null Count	Dtype	
0	DATE FACTURE	47 non-null	object	
1	CODE CLIENT	47 non-null	object	
2	SECTEUR	47 non-null	object	
3	VENDEUR	47 non-null	object	
4	MONTANT	47 non-null	object	
5	Unnamed: 6	0 non-null	float64	

dtypes: float64(1), object(5)
 memory usage: 2.7+ KB

Eliminer les caractères indésirables

- **Difficulté à résoudre :**

- Parfois, dans nos données, on retrouve des **caractères indésirables** qu'il faut **supprimer** ou **remplacer** par d'autres
- Par exemple, dans le fichier « **VentesAgenceU.csv** », colonne « **Montant** » :
 - On retrouve un « _ » dans les chiffres : 1_123,5 → 1123.5
 - Où une « , » à la place du « . » sur les chiffres : 123,5 → 123.5
- On doit « corriger » les données avant de pouvoir les **convertir** aux bons formats (**to_datetime**, **to_numeric**)

- **Solution possible :**

- Appliquer l'opération « **str.replace** »



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Eliminer les caractères indésirables

Colonne où placer les données une fois nettoyées

```
df['MONTANT'] = df['MONTANT'].str.replace('_', '')
```

Considère le contenu de la colonne comme une chaîne de caractères (string)

Ça peut être la même colonne ou pas (nouvelle colonne)...

Caractère à remplacer

Nouveau caractère

Colonne avec les valeurs nettoyées

```
df['MONTANT'] = df['MONTANT'].apply(lambda x: str(x).replace('_', ''))
```

Pour chaque valeur x de la colonne 'Montant', on applique l'action indiquée par la fonction **lambda**

Colonne à modifier

Même chose, mais maintenant en utilisant l'opération **apply**

Caractère à remplacer

Nouveau caractère

On voit la valeur x comme une string

On remplace un caractère par un autre

Eliminer les caractères indésirables



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - Toujours sur le notebook « **ExoExtra1.ipynb** »
 - Observer les données de la colonne « **MONTANT** » avec « **sample** »

`dfFactures.sample(5)`

SECTEUR	VENDEUR	MONTANT
TOMATISME	ENGUENT	7_541,33
TOMATISME	BOUVET	1_459,92
ELECTRICITE	ENGUENT	246,46
TOMATISME	BOUVET	752,98
TOMATISME	BOUVET	198,85

- Remplacer le « `_` » par une chaîne vide « `''` pour les supprimer
- Vérifier le résultat avec « **sample** »

```

dfFactures['MONTANT'] =
    dfFactures['MONTANT'].str.replace('_', '')
dfFactures['MONTANT'].sample(5)

```

CODE FACTURE	MONTANT
FA-2008-0014	917,38
FA-2008-0005	739,83
FA-2008-0034	1051,09
FA-2008-0020	3659,69
FA-2008-0047	204,01

Name: MONTANT, dtype: object

Eliminer les caractères indésirables



- Exemple : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - Remplacer le « , » par un « . »
 - Observer le résultat avec un « **sample** »

```
dfFactures['MONTANT'] =  
    dfFactures['MONTANT'].apply (lambda x: str(x).replace(',', '.',))
```

```
dfFactures['MONTANT'].sample(5)
```

CODE FACTURE	
FA-2008-0012	752.98
FA-2008-0036	485.24
FA-2008-0013	935.47
FA-2008-0024	9367.87
FA-2008-0037	<u>1709.07</u>

Name: MONTANT, dtype: object

Eliminer les caractères indésirables



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - Une fois les données nettoyées, on peut les convertir en « **numérique** » avec l’opération « **to_numeric** » offerte par **Pandas**

```
dfFactures['MONTANT'] =  
    pd.to_numeric(dfFactures['MONTANT'], errors='coerce')  
  
dfFactures.info()
```

Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	DATE FACTURE	47 non-null	object
1	CODE CLIENT	47 non-null	object
2	SECTEUR	47 non-null	object
3	VENDEUR	47 non-null	object
4	MONTANT	47 non-null	float64

L’option **errors = ‘coerce’** remplace par **NaN** les valeurs qu’il n’a pas su convertir. On n’a pas d’erreur dans ce cas, mais un **NaN** à la place.

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- **Obtenir plusieurs informations à partir des dates (jour, mois, année, jour de la semaine, etc.)**
- Transformation des données texte en donnée numérique avec des encodeurs
- Regrouper des données en catégorie (« chaud », « froid », « cher », « abordable », « pas cher »)
- Normalisation des données





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations à partir des dates

• Difficulté à résoudre :

- Parfois, on souhaite obtenir différentes informations à partir d'une date :
 - Jour du mois, année, semaine de l'année, jour de la semaine, etc.

• Solution possible :

- Une fois qu'on a convertit nos données en dates (à l'aide de « `to_datetime` »), on peut extraire des nombreuses informations

Colonne où garder le résultat	Colonne avec les dates	En tant que date (<code>DateTime</code> → <code>dt</code>), on lui demande des informations	Exemple
			2008-04-28
<code>df['JOUR FACTURE'] = df['DATE FACTURE'].dt.day</code>		<i>Jour uniquement</i>	28
<code>df['MOIS FACTURE'] = df['DATE FACTURE'].dt.month</code>		<i>Mois uniquement</i>	4
<code>df['ANNEE FACTURE'] = df['DATE FACTURE'].dt.year</code>		<i>Année uniquement</i>	2008
<code>df['JOUR SEMAINE'] = df['DATE FACTURE'].dt.weekday</code>		<i>Jour de la semaine</i>	0
<code>df['NOM JOUR'] = df['DATE FACTURE'].dt.day_name()</code>		<i>Jour de la sem. (texte)</i>	Monday
<code>df['NOM MOIS'] = df['DATE FACTURE'].dt.month_name()</code>		<i>Mois (texte)</i>	April
<code>df['SEMAINE ANNEE'] = df['DATE FACTURE'].dt.isocalendar().week</code>		<i>Sem. année</i>	18
	<i>Nouvelle version</i>		
	<i>Ancienne version : df['DATE FACTURE'].dt.weekofyear</i>		

Obtenir des informations des dates



- Exemple : colonne « DATE FACTURE » fichier « VentesAgenceU.csv »
 - Toujours sur le notebook « ExoExtra1.ipynb », on va extraire plusieurs informations de la colonne « DATE FACTURE ».
 - Avant cela, on doit convertir la colonne « DATE FACTURE » en date avec l'opération « `to_datetime` »

Colonne avec les valeurs une fois converties

Colonne à convertir

```
dfFactures['DATE FACTURE'] = pd.to_datetime(dfFactures['DATE FACTURE'])

dfFactures['DATE FACTURE'].sample(5)
```

- Un « `sample` » nous permettra d'observer les résultats

CODE FACTURE	
FA-2008-0033	2008-08-14
FA-2008-0036	2008-03-09
FA-2008-0015	2008-04-28
FA-2008-0034	2008-08-23
FA-2008-0041	2008-09-13

Name: DATE FACTURE, dtype: datetime64[ns]

Obtenir des informations des dates



- Exemple : colonne « **DATE FACTURE** » fichier « **VentesAgenceU.csv** »
 - On peut désormais ajouter au *DataFrame* « **dfFactures** » des nouvelles colonnes avec les informations suivantes :
 - Jour → nouvelle colonne « **JOUR FACTURE** »
 - Mois → nouvelle colonne « **MOIS FACTURE** »
 - Année → nouvelle colonne « **ANNEE FACTURE** »
 - Jour de la semaine → nouvelle colonne « **JOUR SEMAINE** »
 - Jour de la semaine (en texte) → nouvelle colonne « **NOM JOUR** »
 - Mois de l'année (en texte) → nouvelle colonne « **NOM MOIS** »
 - Semaine de l'année → nouvelle colonne « **SEMAINE ANNEE** »
 - A nouveau, un « **sample** » nous permettra d'observer les résultats



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates



Nouvelle colonne

Colonne avec la date

```
dfFactures['JOUR FACTURE'] = dfFactures['DATE FACTURE'].dt.day
```

```
dfFactures['MOIS FACTURE'] = dfFactures['DATE FACTURE'].dt.month
```

```
dfFactures['ANNEE FACTURE'] = dfFactures['DATE FACTURE'].dt.year
```

```
dfFactures['JOUR SEMAINE'] = dfFactures['DATE FACTURE'].dt.weekday
```

```
dfFactures['NOM JOUR'] = dfFactures['DATE FACTURE'].dt.day_name()
```

```
dfFactures['NOM MOIS'] = dfFactures['DATE FACTURE'].dt.month_name()
```

Information qu'on souhaite
dt.xxx

Attention aux parenthèses ()

```
dfFactures['SEMAINE ANNEE'] = dfFactures['DATE FACTURE'].dt.isocalendar().week
```

```
dfFactures.sample(5)
```

Pour info

Septembre 2008

	Di	Lu	Ma	Me	Je	Ve	Sa
	1	2	3	4	5	6	
	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30				

ANT

JOUR FACTURE	MOIS FACTURE	ANNEE FACTURE	JOUR SEMAINE	NOM JOUR	NOM MOIS	SEMAINE ANNEE
--------------	--------------	---------------	--------------	----------	----------	---------------

Nouvelles colonnes

FA-2008-0046	2008-09-25	CLI206	AUTOMATISME	ENGUENT	7432.82	1.04	25	9	2008	3	Thursday	September	39
FA-2008-0018	2008-03-06	CLI222	AUTOMATISME	ENGUENT	7432.82	6	3	2008	3	Thursday	March	10	
FA-2008-0022	2008-06-30					30	6	2008	0	Monday	June	27	
FA-2008-0036	2008-03-09					9	3	2008	6	Sunday	March	10	
FA-2008-0044	2008-09-19	CLI206	AUTOMATISME	BOUVET	198.85	19	9	2008	4	Friday	September	38	

Les jours de la semaine sont comptés à partir de 0 pour Lundi
("Monday")

Obtenir des informations à partir des dates

- On peut faire la même chose avec le **temps**, lorsqu'on a des informations temporelles (**date et heure**, ou juste **heure**).
- Par contre, les données doivent contenir, dès le départ, l'information sur l'heure.
– Exemple : la colonne « DATE FACTURE » ne contient pas cette information, les opérations ci-dessous ne marcheront pas sur ces données

*Colonne où garder
le résultat*

```

df['time'] = df['datetime'].dt.time
df['hour'] = df['datetime'].dt.hour
df['minute'] = df['datetime'].dt.minute

```

*Colonne avec l'information
(date / heure)*

2011-12-16 19:42:23+00:00

19:42:23
19
42

*En tant que DateTime (dt), on
lui demande des informations*

*Information demandée
(heure:min:seconde , juste l'heure,
juste les minutes)*

Obtenir des informations des dates



- **Exemple** : colonne « **pickup_datetime** » fichier « **mini_taxi.csv** »
 - Sur le notebook « **ExoExtra2.ipynb** », on va lire le fichier « **mini_taxi.csv** ».
 - Attention à ne pas oublier le « **import pandas as pnd** »
(nouveau notebook, donc à nouveau import)
 - A l'aide d'un « **info** », on remarque la colonne « **pickup_datetime** » est un « **object** ». On va donc devoir la convertir avec « **to_datetime** »

```
import pandas as pnd

dfTaxi = pnd.read_csv(
    'http://kirschpm.fr/cours/PythonDataScience/files/mini_taxi.csv',
    index_col=[0])
```

```
dfTaxi.info()
```

info ↗

#	Column	Non-Null Count	Dtype
0	fare_amount	5999 non-null	float64
1	pickup_datetime	5999 non-null	object
2	pickup_longitude	5999 non-null	float64
3	pickup_latitude	5999 non-null	float64
4	dropoff_longitude	5999 non-null	float64



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates



- Exemple : colonne « pickup_datetime » fichier « mini_taxi.csv »

```
dfTaxi['pickup_datetime'] = pd.to_datetime(dfTaxi['pickup_datetime'])
```

```
dfTaxi.info()
```

Index: 5999 entries, 2009-06-15 17:26:21.000000 to 2014-12-

Data columns (total 7 columns):

#	Column	Non-Null Count	Dtype
0	fare_amount	5999 non-null	float64
1	pickup_datetime	5999 non-null	datetime64[ns, UTC]
2	pickup_longitude	5999 non-null	float64
3	pickup_latitude	5999 non-null	float64
4	dropoff lonaitude	5999 non-null	float64

info

dfTaxi.head()

key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	40.712278	1
2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.979268	40.782004	1
2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.991242	40.750562	2
2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.991567	40.758092	1
2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.956655	40.783762	1

Obtenir des informations des dates



- **Exemple** : colonne « `pickup_datetime` » fichier « `mini_taxi.csv` »

– On va ajouter au *DataFrame* « `dfTaxi` » des nouvelles colonnes avec des informations qu'on pourra extraire des **heures de pickup** :

- **Heure:minute** → nouvelle colonne « `time` »
- **Juste Heure** → nouvelle colonne « `hour` »
- **Juste Minutes** → nouvelle colonne « `time` »

– Un « `sample` » nous permettra d'observer les résultats

```
dfTaxi['time'] = dfTaxi['pickup_datetime'].dt.time
dfTaxi['hour'] = dfTaxi['pickup_datetime'].dt.hour
dfTaxi['minute'] = dfTaxi['pickup_datetime'].dt.minute
```

	fare_amount	pickup_datetime	...	time	hour	minute
key			...			
2010-01-20 20:39:00.0000009	9.7	2010-01-20 20:39:00+00:00	...	20:39:00	20	39
2013-08-09 21:57:46.0000003	7.0	2013-08-09 21:57:46+00:00	...	21:57:46	21	57
2011-09-01 19:26:46.0000003	3.3	2011-09-01 19:26:46+00:00	...	19:26:46	19	26
2010-05-22 23:29:57.0000003	25.3	2010-05-22 23:29:57+00:00		23:29:57	23	29



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates



- **Exemple** : colonne « `pickup_datetime` » fichier « `mini_taxi.csv` »
 - Avec l'information sur **l'heure**, on peut déduire si la course s'est déroulée la **nuit ou non** (ce qui, normalement, aurait une incidence sur le prix)
 - On va utiliser l'opération « `apply` » pour créer une nouvelle colonne « `night` » avec « `True` » si la course s'est déroulée **avant 7h ou après 19h**.

Colonne où garder
le résultat

```
dfTaxi['night'] = dfTaxi['hour'].apply(lambda x: (x<7 or x>19))
```

```
dfTaxi.sample(5)
```

Colonne avec l'heure

Pour **chaque valeur x de la colonne**,
on teste si **x<7 ou si x>19**

La colonne « `night` » contiendra le
résultat de ce test

fare_amount	pickup_datetime	...	time	hour	minute	night
key						
2009-09-08 18:32:47.0000004	9.3 2009-09-08 18:32:47+00:00		18:32:47	18	32	False
2012-04-20 15:01:33.0000004	5.7 2012-04-20 15:01:33+00:00	...	15:01:33	15	1	False
2011-03-05 01:13:04.0000002	9.3 2011-03-05 01:13:04+00:00		01:13:04	1	13	True

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- ✓ Obtenir plusieurs informations à partir des dates
(jour, mois, année, jour de la semaine, etc.)
- **Transformation des données texte en donnée numérique avec des encodeurs**
- Regrouper des données en catégorie (« chaud », « froid »,
« cher », « abordable », « pas cher »)
- Normalisation des données



Transformation des données texte en numérique

- **Difficulté à résoudre :**

- Lorsque certaines **colonnes** contiennent des **textes** (des « **catégories** »), il faut les « **traduire** » en **valeurs numériques**
- Exemples :
 - Genre : « male » / « female »,
 - Navigateurs : « Safari » / « Chrome » / « Firefox »
- Les différents modèles de ML proposés sur **SKLearn ne savent pas** travailler avec de texte

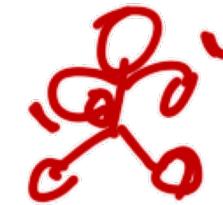
- **Solution possible :**

- Utiliser des « **encodeurs** »
 - Les encodeurs **traduisent** des **catégories** exprimées en texte en format **numérique**
- Différents types de ces « **encodeurs** » existent
 - Proposés par la bibliothèque **Sklearn**, mais aussi par la bibliothèque **Pandas**

Transformation des données texte en numérique

- Transformation des données en **Sklearn**
 - SKLearn ne manipule que des données numériques
 - On considère que le **texte** indique des **catégories**
 - Pas un texte « libre »

['approves' , 'disapproves']  [0 1]



• Encoders sur Sklearn

- Différents **encoders** disponibles sur **sklearn.preprocessing**
 - **LabelEncoder** : Transformation des **labels** (**target**) en valeurs entières (0 à n-1)
 - **OrdinalEncoder** : Transformation des **données** (**features**) en valeurs entières (de 0 à n-1)
 - **OneHotEncoder** : Transformation des **données** (**features**) en valeurs **binaires**
- Quelque soit l'encoder, on suit les mêmes **étapes** :
Création encodeur → entraînement (« fit ») → application (« transform »)



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	approves

Target (classes)
« Y_set »

Y = labEnc.inverse_transform(Yenc)

Transformation inverse
(des valeurs aux labels)

['approves' 'disapproves' 'approves' 'approves' 'disapproves' 'disapproves'

'approves' 'approves'] ← Y_set : valeurs originales

[0 1 0 0 1 1 0 0] ← Y_enc : valeurs encodées

LabelEncoder

Conversion des targets en valeurs numériques

```
from sklearn.preprocessing import LabelEncoder
```

```
labEnc = LabelEncoder()  
labEnc.fit( Y_set )
```

Création et
entraînement de
l'encoder

Yenc = labEnc.transform(Y_set)

Transformation
des valeurs

valeurs

labEnc.classes_ ← Classes retrouvées

['approves' 'disapproves']

Transformation de texte en numérique



- **Exemple :** transformation d'un « **target** » texte en numérique

- Sur le nouveau notebook « **ExoExtra3.ipynb** »,
 on va créer un DataFrame simple,
 juste pour nos tests

```
import pandas as pd

dfCategories = pd.DataFrame (
    [ ['male', 'from US', 'uses Safari', 'approves'],
    ['female', 'from Europe', 'uses Firefox', 'disapproves'] ,
    ['female', 'from US', 'uses Safari', 'approves'],
    ['male', 'from Europe', 'uses Safari', 'approves'],
    ['female', 'from US', 'uses Firefox', 'disapproves'] ,
    ['male', 'from Europe', 'uses Chrome', 'disapproves'] ,
    ['female', 'from Asia', 'uses Chrome', 'approves'],
    ['male', 'from Asia', 'uses Chrome', 'approves'] ],
    columns=['sex', 'region','browser', 'vote' ] )
```

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	approves

dfCategories

Transformation de texte en numérique



- **Exemple** : transformation d'un « **target** » texte en numérique

- On va créer maintenant un **LabelEncoder**
- On entraîne l'encoder avec la colonne « **vote** »

```
from sklearn.preprocessing import LabelEncoder
```

Ne pas oublier l'import

```
labEnc = LabelEncoder()
```

Création de l'objet encoder

```
labEnc.fit( dfCategories[ 'vote' ] )
```

*Entrainement de l'encoder avec la colonne
contenant les catégories qu'on souhaite
transformer.*

*On peut voir les **classes** qu'il a reconnu (après le **fit**)*

```
print (labEnc.classes_)
```

*La valeur « approves »
sera transformée en 0*

['approves' 'disapproves']

0

*« disapproves »
en 1*



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple :** transformation d'un « **target** » texte en numérique
 - Maintenant, on peut utiliser l'encoder pour transformer la colonne « **vote** »
 - On va créer une nouvelle colonne « **code_vote** » avec les valeurs transformées

Colonne qui va recevoir les valeurs transformées

Colonne à transformer

Transformation des données selon le modèle d'encoder entraîné

```
dfCategories['code_vote'] = labEnc.transform( dfCategories['vote'] )
```

dfCategories

	sex	region	browser	vote	code_vote
0	male	from US	uses Safari	approves	0
1	female	from Europe	uses Firefox	disapproves	1
2	female	from US	uses Safari	approves	0
3	male	from Europe	uses Safari	approves	0
4	female	from US	uses Firefox	disapproves	1
5	male	from Europe	uses Chrome	disapproves	1
6	female	from Asia	uses Chrome	approves	0
7	male	from Asia	uses Chrome	approves	0

Nouvelle colonne avec les valeurs transformées

Transformation de texte en numérique



- **Exemple** : transformation d'un « **target** » texte en numérique

- On peut procéder à la **transformation inverse** d'un *array* de valeurs et obtenir les catégories correspondantes
- On va créer d'abord un *array* avec une séquence aléatoire de 0 et 1

```
import numpy as np
```

On va utiliser la bibliothèque Numpy pour générer l'array de n's aléatoires

```
aleatoire = np.random.randint(low=0, high=2, size=5)
```

```
print (aleatoire)
```

[0 1 0 1 0]

Randint produit un array de taille « **size** » (5) avec des chiffres entre « **low** » (0) et « **high** » (2), celui-ci exclut (donc de 0 et 1)

```
cate_aleatoire = labEnc.inverse_transform(aleatoire)
```

```
print (cate_aleatoire)
```

['approves' 'disapproves' 'approves' 'disapproves' 'approves']

On réalise la transformation inverse

On obtient les classes correspondantes



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	

Features (variables)
« X_set »

L'encoder attend
un **DataFrame**
avec un ensemble
de **features**

X_set : valeurs originales

```
['male' 'from Europe' 'uses Safari']
['female' 'from US' 'uses Firefox']
['male' 'from Europe' 'uses Chrome']
['female' 'from Asia' 'uses Chrome']
```

Xord : valeurs encodées

OrdinalEncoder

Conversion des **features** en valeurs **entiers**

```
from sklearn.preprocessing import OrdinalEncoder
```

```
ordEnc = OrdinalEncoder()
ordEnc.fit( X_set )
```

Création et
entraînement de
l'encoder

```
Xord = ordEnc.transform( X_set )
```

Transformation
des valeurs

valeurs

```
X = ordEnc.inverse_transform(Xord)
```

Transformation inverse
(des valeurs aux données)

[1.	1.	2.]
[0.	2.	1.]
[1.	1.	0.]
[0.	0.	0.]

```
ordEnc.categories_
```

Liste des
catégories

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique

- On va désormais utiliser un **OrdinalEncoder**, qu'on va créer
- On entraîne l'encoder avec un **DataFrame** ne contenant que colonne « **sex** »

```
from sklearn.preprocessing import OrdinalEncoder
Ne pas oublier l'import
```

```
ordEnc = OrdinalEncoder()
Création de l'objet encoder
```

```
ordEnc.fit( dfCategories[['sex']] )
Entrainement de l'encoder
Attention à l'usage des [[ ]], car les
encoders pour les features attendent tout
un DataFrame et pas une simple colonne
(comme le LabelEncoder)
```

On peut voir les **catégories** qu'il a reconnu (*après le fit*)

```
print (ordEnc.categories_)
```

```
[array(['female', 'male'], dtype=object)]
```

La valeur « female » sera transformée en 0

0

1

« male » sera transformé en 1



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique
 - On peut maintenant utiliser un **OrdinalEncoder** pour transformer les valeurs de la colonne « **sex** »
 - On va ajouter une nouvelle colonne « **code_sex** » avec les valeurs transformées

*Colonne qui va recevoir
les valeurs transformées*

```
dfCategories['code_sex'] = ordEnc.transform( dfCategories [['sex']] )
```

dfCategories

	sex	region	browser	vote	code_vote	code_sex
0	male	from US	uses Safari	approves	0	1.0
1	female	from Europe	uses Firefox	disapproves	1	0.0
2	female	from US	uses Safari	approves	0	0.0
3	male	from Europe	uses Safari	approves	0	1.0
4	female	from US	uses Firefox	disapproves	1	0.0
5	male	from Europe	uses Chrome	disapproves	1	1.0
6	female	from Asia	uses Chrome	approves	0	0.0
7	male	from Asia	uses Chrome	approves	0	1.0

*DataFrame contenant les **features**
à transformer.*

*Attention donc à l'usage des **[[]]**
afin d'avoir un **DataFrame** avec les
colonnes qu'on veut **transformer***

*Nouvelle colonne avec
les valeurs transformées*

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique

- Pour illustrer la **transformation inverse**, on va utiliser les valeurs de notre *array* de valeurs 0 et 1 aléatoires
- Il faut d'abord créer un *DataFrame* à partir de cet *array*

```
dfAleatoire = pnd.DataFrame(aleatoire)
print (dfAleatoire)

ordEnc.inverse_transform( dfAleatoire )
```

```
print(dfAleatoire)
```

	0
0	1
1	0
2	1
3	0
4	1

Données dfAleatoire

```
array([['male'],
       ['female'],
       ['male'],
       ['female'],
       ['male']], dtype=object)
```

On va utiliser la variable « aleatoire » qu'on avait créé dans l'exercice précédent et construire un DataFrame avec (puisque OrdinalEncoder attend un DataFrame avec des valeurs)

Catégories obtenues avec la transformation inverse



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
-				



Chaque colonne sera « éclatée » en plusieurs, en fonction du nombre de catégories présentes.

sex	region	browser
[0. 1.]	[0. 0. 1.]	[0. 0. 1.]
[1. 0.]	[0. 1. 0.]	[0. 1. 0.]
[1. 0.]	[0. 0. 1.]	[0. 0. 1.]

```
print(Xoh.toarray())
```

OneHotEncoder

Conversion des **features** en valeurs **binaires**

```
from sklearn.preprocessing import OneHotEncoder
```

```
ohEnc = OneHotEncoder()
ohEnc.fit( X_set )
```

Création et entraînement de l'encoder

```
Xoh = ohEnc.transform( X_set )
```

Transformation des valeurs

DataFrame avec les valeurs

```
X = ohEnc.inverse_transform(Xoh)
```

Transformation inverse (des valeurs aux données)

```
ohEnc.get_feature_names()
```

Liste des catégories

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique
 - On va utiliser maintenant un **OneHotEncoder** pour encoder un **DataFrame** avec la colonne « **region** ».

```
from sklearn.preprocessing import OneHotEncoder
```

Ne pas oublier l'import

```
ohEnc = OneHotEncoder()
```

Création de l'objet encoder

```
ohEnc.fit( dfCategories[['region']] )
```

Entrainement de l'encoder
Attention à l'usage des [[]], car les encoders pour les features attendent tout un DataFrame et pas une simple colonne (comme le LabelEncoder)

On peut voir les catégories qu'il a reconnu (après le fit)

```
print (ohEnc.get_feature_names())
```

```
['x0_from Asia' 'x0_from Europe' 'x0_from US']
```

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique
 - Une fois entraîné, on peut transformer les valeurs présentes dans la colonne « **region** ».

```
Xoh = ohEnc.transform( dfCategories[['region']] )
```

```
print ( Xoh.toarray() )
```

On va créer un **nouveau DataFrame** avec ces valeurs afin de pouvoir les ajouter à notre DataFrame original

	x0_from Asia	x0_from Europe	x0_from US
0	0.0	0.0	1.0
1	0.0	1.0	0.0
2	0.0	0.0	1.0
3	0.0	1.0	0.0
4	0.0	0.0	1.0
5	0.0	1.0	0.0
6	1.0	0.0	0.0
7	1.0	0.0	0.0

DataFrame avec les valeurs à transformer

*Les valeurs transformées sont dans un tableau, avec autant des **colonnes** que des **catégories** retrouvées.*

```
[[0. 0. 1.]  

 [0. 1. 0.]  

 [0. 0. 1.]  

 [0. 1. 0.]  

 [0. 0. 1.]  

 [0. 1. 0.]  

 [1. 0. 0.]  

 [1. 0. 0.]]
```

```
dfXoh = pnd.DataFrame(Xoh.toarray() ,  

columns=ohEnc.get_feature_names())
```

dfXoh



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



• Exemple : transformation d'une « feature » texte en numérique

- Puis, on peut ajouter les nouvelles valeurs transformées à notre DataFrame « **dfCatégories** » (sur un nouveau DataFrame ou sur le même).

```
dfCategories_OneHot = pd.concat( [ dfCategories, dfXoh ] ,  
                                 axis='columns' )
```

Avec l'opération **concat** de Pandas
ou

Avec l'opération **join** propre aux **DataFrame**

Deux manières de combiner un DataFrame à un autre :

```
dfCategories_OneHot = dfCategories.join(dfXoh)
```

dfCategories_OneHot

	sex	region	browser	vote	code_vote	code_sex	x0_from Asia	x0_from Europe	x0_from US
0	male	from US	uses Safari	approves	0	1.0	0.0	0.0	1.0
1	female	from Europe	uses Firefox	disapproves	1	0.0	0.0	1.0	0.0
2	female	from US	uses Safari	approves	0	0.0	0.0	0.0	1.0
3	male	from Europe	uses Safari	approves	0	1.0	0.0	1.0	0.0
4	female	from US	uses Firefox	disapproves	1	0.0	0.0	0.0	1.0

Transformation des données texte en numérique

- **Encoding - Bibliothèque Pandas**

- La bibliothèque **Pandas** propose un *encoding* de type **One Hot** appelé « **get_dummies** »
- Pas besoin de phase d'entraînement, on l'utilise directement

df_weather.value_counts(df_weather['Description'])		Description
		Normal 4992
		Warm 2507
		Cold 2501
		dtype: int64

On a 3 **catégories** (« Normal », « Warm » et « Cold »), chacune deviendra une nouvelle **colonne**.

Description		Description_Cold	Description_Normal	Description_Warm
Cold		1	0	0
Warm	pnd.get_dummies (df_weather)	0	0	1
Normal		0	1	0
Cold		1	0	0
Cold		1	0	0





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

• Encoding - Bibliothèque Pandas

- On peut réduire le nombre de colonnes créées avec l'option « *drop_first = True* »

```
df_dummies = pd.get_dummies ( df_weather, drop_first=True )
```

Dans **OneHotEncoder**, on écrit
OneHotEncoder(drop='first')

On représente la 1^{ère} catégorie par la combinaison de 0 sur les autres

On obtient 1 colonne en moins

```
Data columns (total 9 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --  
 0   Temperature_c    10000 non-null   float64  
 1   Humidity          10000 non-null   float64  
 2   Wind_Speed_kmh   10000 non-null   float64  
 3   Wind_Bearing_degrees 10000 non-null int64  
 4   Visibility_km     10000 non-null   float64  
 5   Pressure_millibars 10000 non-null   float64  
 6   Rain              10000 non-null   int64  
 7   Description_Normal 10000 non-null   uint8  
 8   Description_Warm   10000 non-null   uint8
```

Nouvelles colonnes

Là où on avait « 1 » sur la colonne « **Description_cold** », on se retrouve juste avec « 0 » sur les autres colonnes.

	Description_Normal	Description_Warm
0	cold	0
1	0	1
2	1	0
3	0	0
4	0	0

df_dummies.head()

Transformation de texte en numérique



- **Exemple** : transformation d'une « **feature** » texte en numérique
 - Toujours sur le notebook « **ExoExtra3.ipynb** », on va utiliser l'encoder « **get_dummies** » pour transformer les colonnes « **region** » et « **browser** ».

Si on a bien fait

`import pandas as pd`
 au début de notre notebook

```
dummies = pd.get_dummies (dfCategories [['region', 'browser']],
                           drop_first=True)
```

`dummies`

DataFrame avec les colonnes à transformer (notre « X set »)

*Option **drop_first** pour réduire le nombre de colonnes*

	region_from Europe	region_from US	browser_uses Firefox	browser_uses Safari
0	0	1	0	1
1	1	0	1	0
2	0	1	0	1
3	1	0	0	1
4	0	1	1	0
5	1	0	0	0
6	0	0	0	0
7	0	0	0	0



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : transformation d'une « **feature** » texte en numérique
 - Une fois les valeurs transformées, on peut les ajouter à notre DataFrame

DataFrame original et DataFrame avec les données transformées

```
dfCategories_dummies = pd.concat ( [ dfCategories, dummies ],  
                                 axis='columns' )
```

DataFrame avec toutes les données (originelles et transformées)

	sex	region	browser	vote	code_vote	code_sex	region_from Europe	region_from US	browser_uses Firefox	browser_uses Safari
0	male	from US	uses Safari	approves	0	1.0	0	1	0	1
1	female	from Europe	uses Firefox	disapproves	1	0.0	1	0	1	0
2	female	from US	uses Safari	approves	0	0.0	0	1	0	1
3	male	from Europe	uses Safari	approves	0	1.0	1	0	0	1
4	female	from US	uses Firefox	disapproves	1	0.0	0	1	1	0
5	male	from Europe	uses Chrome	disapproves	1	1.0	1	0	0	0
6	female	from Asia	uses Chrome	approves	0	0.0	0	0	0	0
7	male	from Asia	uses Chrome	approves	0	1.0	0	0	0	0

Transformation des données texte en numérique

- Pourquoi utiliser « **OneHotEncoder** » (ou « **get_dummies** ») plutôt qu'un « **OrdinalEncoder** » ?



– **OrdinalEncoder** transforme les catégories dans une **séquence des valeurs**

- « uses Safari » → 0, « uses Firefox » → 1, « uses Chrome » → 2

- Cette séquence de valeurs va être interprétée comme telle par les algorithmes de Machine Learning

– Cette interprétation peut induire à des conclusions erronées

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- ✓ Obtenir plusieurs informations à partir des dates
(jour, mois, année, jour de la semaine, etc.)
- ✓ Transformation des données texte en donnée numérique
avec des encodeurs
- **Regrouper des données en catégorie**
- Normalisation des données



Regrouper des données en catégorie

- **Difficulté à résoudre :**

- Parfois, on souhaite regrouper nos données en catégories
 - Par exemple : au lieu d'avoir des valeurs d'âge (6, 20, 56...), avoir des classes d'âge (enfant, adulte, senior...)

- **Solution possible :**

- Transformer une variable quantitative (âge) en variable qualitative (classe d'âge)
- On appelle ça faire de la « **discrétisation** »
- On peut utiliser les opérations « **cut** » et « **qcut** » proposées par **Pandas** pour trouver des catégories à partir des données
 - Opération « **cut** » : organise les données en ***n*** catégories (on fournit ***n***)
 - Opération « **qcut** » : organise les données en ***n*** catégories et s'assure que celles-ci sont **équilibrées** (avec une **répartition constante** d'individus)



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Regrouper des données en catégorie

- Opérations « **cut** » et « **qcut** » : plusieurs possibilités

Colonne qui recevra les catégories

```
df['categorie']= pnd.cut ( df['prix'], bins=4 )
```

Colonne avec les données à discréteriser

```
df['categorie'] = pnd.cut (df['prix'], bins=4,
```

Nombre de catégories à utiliser

```
labels=['pas cher', 'normal',  
'cher', 'très cher'])
```

Étiquettes pour les catégories

```
df['categorie'] = pnd.cut (df['prix'], bins=[ df['prix'].min(),
```

```
30, 50, 100,  
df['prix'].max() ] )
```

Tranches pour les catégories

min-30 , 30-50, 50-100, 100-max

Option **labels** aussi possible avec **qcut**

```
df['qcategorie'] = pnd.qcut( df['prix'] , q=4 )
```

Colonne qui recevra les catégories

Colonne avec les données

Nombre de catégories souhaitées

Regrouper des données en catégorie



- **Exemple** : extraire des catégories de prix

- Sur un nouveau notebook « **ExoExtra4.ipynb** », on va lire le fichier « **mini_taxi.csv** ».
 - Attention à ne pas oublier le « **import pandas as pd** » (**nouveau notebook → import**)
- On va essayer d'extraire des catégories de prix à partir de la colonne « **fare_amount** »

```
import pandas as pd

dfTaxi =
pd.read_csv('http://kirschpm.fr/cours/PythonDataScience/files/mini_taxi.csv',
            index_col=[0])

dfTaxi.info()
```

Index: 5999 entries, 2009-06-15 17:26:21.000000
Data columns (total 7 columns):

#	Column	Non-Null Count	Dtype
0	fare_amount	5999 non-null	float64
1	pickup_datetime	5999 non-null	object
2	pickup_longitude	5999 non-null	float64
3	pickup_latitude	5999 non-null	float64

Regrouper des données en catégorie



- **Exemple** : extraire des catégories de prix

— Afin de mieux comprendre les valeurs qu'on retrouve dans la colonne « **fare_amount** », on va utiliser l'opération « **describe** »

```
dfTaxi['fare_amount'].describe()
```

count	5999.000000
mean	11.385616
std	9.805378
min	-2.900000
25%	6.000000
50%	8.500000
75%	12.900000
max	180.000000

Name: fare_amount, dtype: float64

Valeurs négatives

```
dfTaxi.loc[dfTaxi['fare_amount'] < 0]
```

	fare_amount	pickup_datetime	pickup_lo
key			
2010-03-09 23:37:10.0000005	-2.9	2010-03-09 23:37:10 UTC	-73.
2015-03-22 05:14:27.0000001	-2.5	2015-03-22 05:14:27 UTC	-74



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



• Exemple : extraire des catégories de prix

- On ne garde que les lignes avec un prix « correct » (c.a.d. $\text{prix} \geq 0$)

On remplace **dfTaxi** par l'ensemble de valeurs qui vérifient la **condition**

```
dfTaxi = dfTaxi.loc[ dfTaxi['fare_amount'] >= 0 ]
```

```
dfTaxi['fare_amount'].describe()
```

```
dfTaxi.sample(5)
```

count	5997.000000	Moins de valeurs dans le DataFrame
mean	11.390313	
std	9.803637	
min	0.010000	Plus de valeur négative
25%	6.000000	
50%	8.500000	
75%	12.900000	
max	180.000000	
Name: fare_amount, dtype: float64		

key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
2009-02-14 08:13:00.000000022	8.1	2009-02-14 08:13:00 UTC	-73.984345	40.712874
2010-12-18 02:34:31.00000003	6.5	2010-12-18 02:34:31 UTC	-73.984948	40.712874
2010-01-11 15:42:00.0000000186	7.3	2010-01-11 15:42:00 UTC	-73.981355	40.712874
2015-02-03 08:02:33.00000003	4.5	2015-02-03 08:02:33 UTC	-73.988861	40.712874
2015-03-22 16:37:27.00000003	4.5	2015-03-22 16:37:27 UTC	-73.981056	40.712874

Transformation de texte en numérique

ExoExtra4.ipynb



- **Exemple** : extraire des catégories de prix

- Maintenant qu'on a des données propres, on peut organiser les prix (colonne « **fare_amount** ») en 4 catégories
- On créer 2 nouvelles colonnes dans le *DataFrame* **dfTaxi**
 - Colonne « **cat_prix** » avec les catégories
 - Colonne « **cat_prix_label** » avec les catégories en label

```
dfTaxi['cat_prix']= pnd.cut (dfTaxi['fare_amount'], bins=4)
```

```
dfTaxi['cat_prix_label'] = pnd.cut (dfTaxi['fare_amount'], bins=4,
                                     labels=['pas cher', 'normal', 'cher', 'très cher'])
```

```
dfTaxi.sample(5)
```

	fare_amount	pickup_datetime		passenger_count	cat_prix	cat_prix_label
key			...			
2013-08-31 15:49:00.00000034	6.0	2013-08-31 15:49:00 UTC	...	1	(-0.17, 45.008]	pas cher
2015-02-27 07:50:56.0000003	10.5	2015-02-27 07:50:56 UTC	...	1	(-0.17, 45.008]	pas cher
2015-05-19 06:43:30.0000006	11.0	2015-05-19 06:43:30 UTC	...	2	(-0.17, 45.008]	pas cher



- **Exemple** : extraire des catégories de prix
 - A l'aide de l'opération « `value_counts` », on peut vérifier la distribution des individus dans les classes
 - On observe que cette distribution n'est pas homogène, vu que la première catégorie (jusqu'à \$45) regroupe une majorité d'individus

```
print (dfTaxi['cat_prix'].value_counts())
```

(-0.17, 45.008]	5876
(45.008, 90.005]	118
(135.002, 180.0]	2
(90.005, 135.002]	1

*Catégories très déséquilibrées,
ce qui peut poser problème pour
les modèles de ML*

Transformation de texte en numérique



- **Exemple** : extraire des catégories de prix

- L'opération « **qcut** » permet alors d'obtenir des catégories équilibrées, c.a.d. avec une répartition uniforme des individus entre les catégories
- On va l'utiliser pour faire une nouvelle répartition des valeurs de la colonne « **fare_amount** »

```
dfTaxi['qcat_prix'] = pnd.qcut( dfTaxi['fare_amount'], q=4 )
```

```
print (dfTaxi['qcat_prix'].value_counts())
dfTaxi.sample(5)
```

	(6.0, 8.5]	1594
	(0.0090000000000001, 6.0]	1542
	(12.9, 180.0]	1470
	(8.5, 12.9]	1391
	Name: qcat_prix, dtype: int64	

fare_amount	cat_prix	cat_prix_label	qcat_prix
key			
2012-08-11 17:31:14.0000004	(-0.17, 45.008]	pas cher	(6.0, 8.5]
2010-11-27 01:51:28.0000007	(-0.17, 45.008]	pas cher	(6.0, 8.5]
2014-06-10 14:18:29.0000001	(-0.17, 45.008]	pas cher	(8.5, 12.9]

Catégories équilibrées
Individus répartis de manière équilibrée entre les catégories

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- ✓ Obtenir plusieurs informations à partir des dates
(jour, mois, année, jour de la semaine, etc.)
- ✓ Transformation des données texte en donnée numérique
avec des encodeurs
- ✓ Regrouper des données en catégorie
- **Normalisation des données**



Normalisation

ExempleScaler.ipynb



• Normalisation

- Certains algorithmes vont être particulièrement sensibles à l'**ordre de grandeur des variables**
 - Régression linéaire, K-Means...
- Lorsque celle-ci est trop différente, une **normalisation** peut s'avérer nécessaire
 - Ex : prix des maisons (500 000 €) et taux bancaires (0,05)
- Différent types de « **scalers** » existent
 - Paquet **sklearn.preprocessing**
 - **StandardScaler** : normalisation basée sur la moyenne
 - **MinMaxScaler** : normalisation à partir des valeurs de min et max
 - **RobustScaler** : normalisation plus « robuste » (moins sensible aux *outliers*), basée sur la moyenne

Normalisation

ExempleScaler.ipynb



• Normalisation

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
x_train_scal = scaler.fit_transform(x_train)
```

```
x_test_scal = scaler.transform(x_test)
```

On entraîne le Scaler avec les données d'entraînement

On transforme les données de test avant de les utiliser

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
```

```
x_train_scal = scaler.fit_transform(x_train)
```

```
x_test_scal = scaler.transform(x_test)
```

On peut indiquer au **MinMaxScaler** l'intervalle des données à utiliser pour la normalisation.

Normalisation

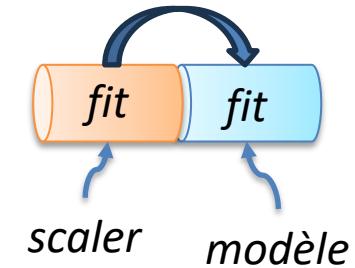
ExempleScaler.ipynb



• Pipeline

- Puisque les données devront être systématiquement transformées avant d'être utilisées, on peut définir un « **pipeline** » de transformation
- Dans le pipeline, les données sont transmises d'une opération à une autre, dans une chaîne :
 - La sortie du **fit** du **scaler** est transmise au **fit** du **modèle**
 - La sortie du **transform** du **scaler** est transmise au **predict** du **modèle**

```
scaler = MinMaxScaler()  
  
linear = LinearRegression()  
  
pipeline = Pipeline (  
    [ ('minmax', scaler),  
     ('linear', linear) ] )
```



Plus besoin de transformer les données avant, le pipeline s'en occupe

```
pipeline.fit(x_train, y_train)  
y_pred = pipeline.predict(x_test)
```