



Régression

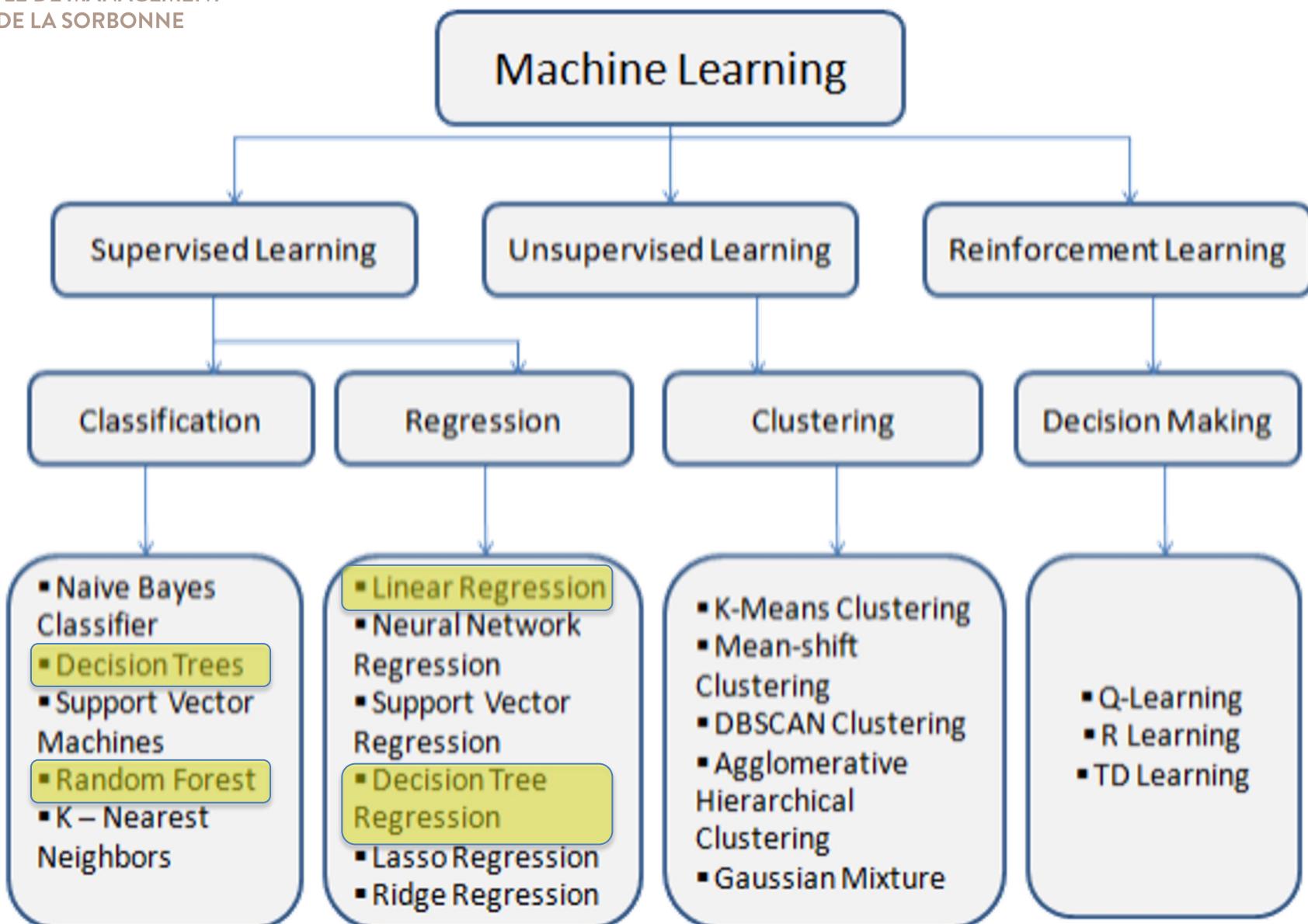
Arbres de régression Régression linéaire

Fichiers sur

<https://github.com/mkirschpin/CoursPython>

<http://kirschpm.fr/cours/PythonDataScience/>

Types of Machine Learning





UNIVERSITÉ PARIS 1

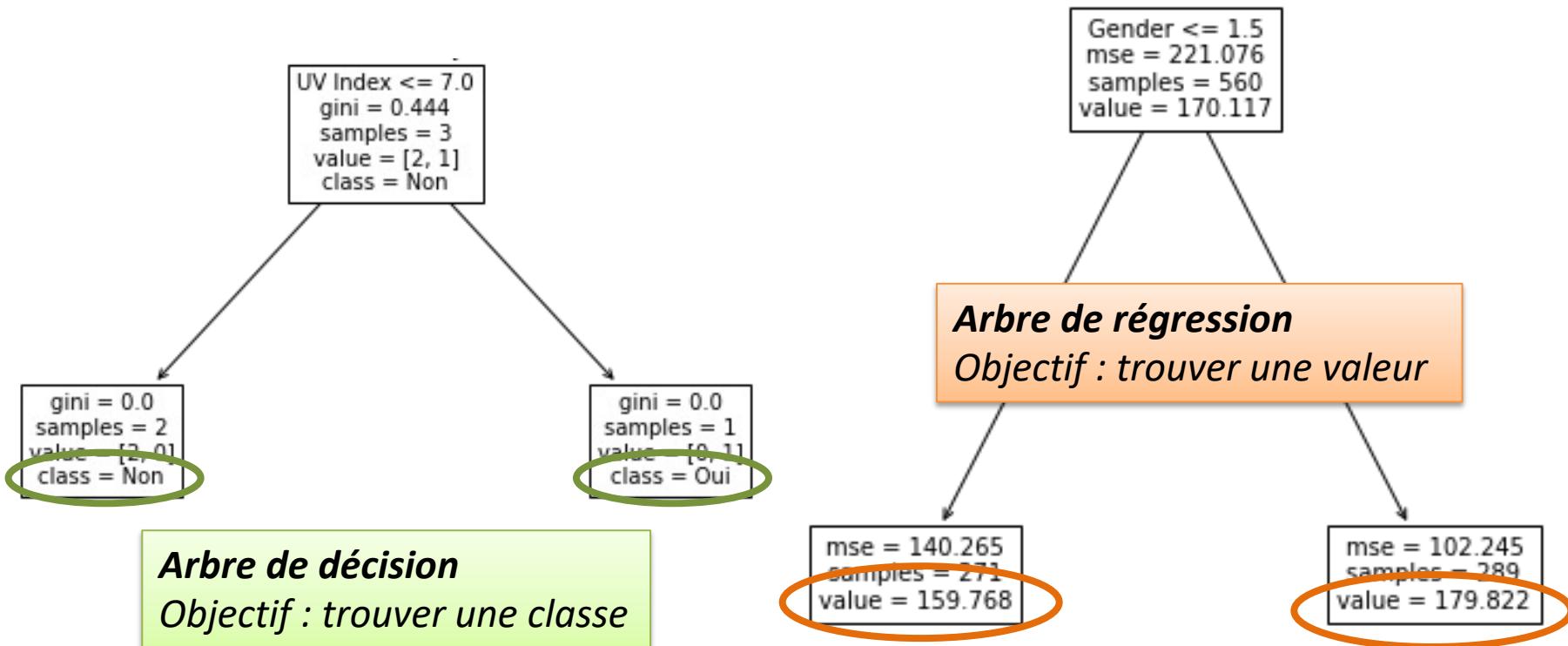
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Arbres de Régression

- **Arbres de Régression**

- Objectif : trouver une valeur (**quantitative**) au lieu d'une classe (**qualitative**)



Arbres de Régression

- Bases :
 - Mesures pour **regrouper les individus** dans les nœuds et **former l'arbre**
 - On cherche à minimiser l'erreur entre la valeur prédite et la valeur réelle de **l'ensemble d'entraînement**

MAE (L1)

Mean Absolute Error

A partir de la **médiane**
Différence **absolue** entre
les valeurs et la médiane

MSE (L2)

Mean Squared Error

A partir de la **moyenne**
Différence au **carré** entre
les valeurs et la moyenne



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Arbres de Régression

- Attention :
 - Le choix du critère (=mesure) retenu pour **regrouper les individus** dans les nœuds doit être pris en compte pour le choix de la **métrique d'évaluation** !
 - Si $h(x_i)$ est la prédiction pour la variable x_i et y_i la valeur réelle dans l'ensemble de test, la mesure de l'écart sera différente selon la métrique MAE et la métrique MSE :

$$mae = \frac{1}{m} \left(\sum_{i=1}^m |h(x_i) - y_i| \right)$$

$$mse = \frac{1}{m} \left(\sum_{i=1}^m (h(x_i) - y_i)^2 \right)$$

$h(x_i)$ → *Prédiction pour la valeur x_i*

$$rmse = \sqrt[2]{mse}$$



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Arbres de Régression

```
from sklearn.tree import DecisionTreeRegressor
```

Création objet modèle

```
rgr = DecisionTreeRegressor(criterion='mae')
```

selon la version...

'mae' → 'absolute_error'

```
criterion='mse'
```

'mse' → 'squared_error'

*On entraîne le modèle (**fit**) et
on l'utilise pour faire les
prédictions (**predict**)*

```
rgr.fit(x_train, y_train)  
rgr.predict(x_test)
```

*On choisit le critère de construction de l'arbre. On peut aussi choisir sa profondeur (**max_depth=n**)*

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
mse = mean_squared_error(y_test, y_test_pred)
```

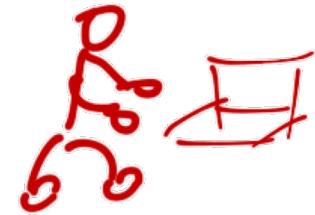
*On évalue les résultats avec
la métrique choisie*

```
mae = mean_absolute_error(y_test, y_test_pred)
```

*Valeurs attendues
(celles de l'ensemble de test)*

Valeurs prédictes par le modèle

Hands On !



- Exercice : prévoir la taille de quelqu'un à partir de son genre
 - Créer un nouveau Notebook...
 - Créer un **DataFrame** avec des valeurs ± aléatoires
 - On va créer 2 populations de 200 personnes chacune :
 - 1 population de femmes petites et 1 d'hommes petits

On va utiliser la bibliothèque NumPy pour les n° aléatoires

```
import numpy as np
import pandas as pd
```

n = 200

```
height_pop1_f = np.random.normal( loc=155, scale=4, size=n )
height_pop1_m = np.random.normal( loc=175, scale=5, size=n )
```

*L'opération **random.normal** produit un array de numéros aléatoires suivant une distribution normale (Gaussienne)*

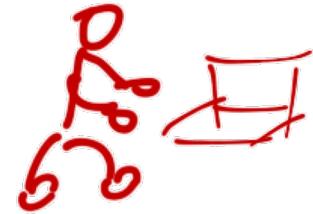
loc= centre de la distribution (moyenne)

scale= écart type pour la distribution

size= nombre d'éléments

- Même chose pour les personnes plus grandes (**loc=165/185, scale=15/12**)
 - On crée **height_pop2_f** et **height_pop2_m**

Hands On !



- Exercice : prévoir la taille de quelqu'un à partir du genre
 - On réunit toutes les femmes dans un seul **Array** (idem pour les hommes)

*L'opération **concatenate** permet de « coller » une liste d'arrays dans un nouvel array.*

```
height_f = np.concatenate([ height_pop1_f, height_pop2_f ] )
```

Array valeurs femmes

```
height_m = np.concatenate([ height_pop1_m, height_pop2_m ] )
```

Array valeurs hommes

- On réunit les populations dans un seul **DataFrame**

- En ajoutant le genre

```
import pandas as pd
```

```
df_height = pd.DataFrame ( {
    'Gender': [1 for i in range(height_f.size)] + ... et height_h.size valeurs
                [2 for i in range(height_m.size)], « 2 » (les hommes)
    'Height': np.concatenate([ height_f, height_m ])
}
```

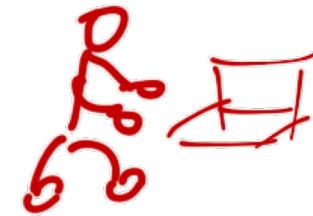
*Fait une liste avec **height_f.size** valeurs « 1 » (les femmes)...*

*... et **height_h.size** valeurs « 2 » (les hommes)*

*On « colle » toutes les valeurs (femmes et hommes) dans la colonne **Height***



Hands On !



- Exercice : prévoir la taille de quelqu'un à partir du genre
 - On vérifie le DataFrame créé

`df_height.sample(5)`

	Gender	Height
383	1	140.982568
311	1	148.684059
677	2	178.690035
626	2	184.212748
463	2	175.683897

L'opération **sample** permet d'afficher **n** éléments (choisis au hasard)

`df_height.describe()`

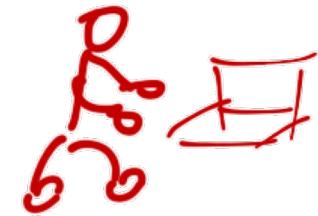
	Gender	Height
count	800.000000	800.000000
mean	1.500000	170.187320
std	0.500313	15.495558
min	1.000000	120.497399
25%	1.000000	157.073551
50%	1.500000	170.954099
75%	2.000000	180.236043
max	2.000000	219.315758

`df_height.groupby(by='Gender').agg(['mean', 'median', 'max', 'min'])`

L'expression **groupby(...).agg(...)** permet d'agréger une liste d'opérations (*mean*, *max*...) sur un groupe

Gender	Height			
	mean	median	max	min
1	159.716407	157.064947	215.826414	120.497399
2	180.658232	179.085719	219.315758	158.060598

Hands On !



- Exercice : prévoir la taille de quelqu'un à partir du genre**
 - On peut visualiser la distribution des données

```
%matplotlib inline
import matplotlib.pyplot as plt
```

Création d'une figure avec de l'espace (ax) pour un seul graphique (1,1 → 1 ligne, 1 colonne)

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
```

Plot valeurs femmes

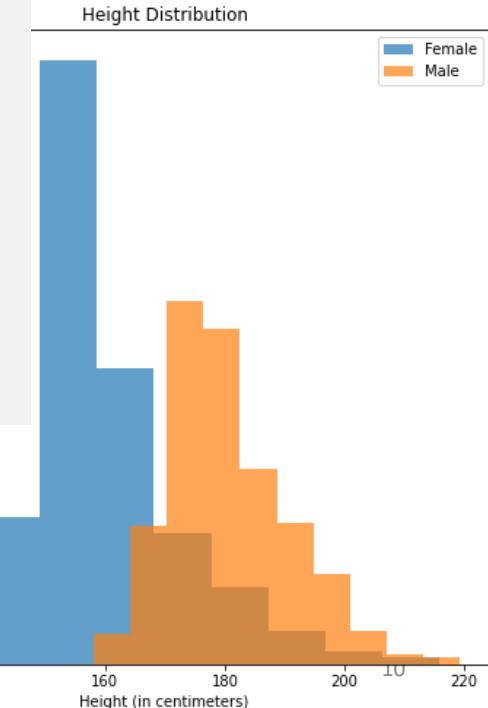
```
df_height[df_height['Gender'] == 1]['Height'].plot (
    label='Female', kind='hist', alpha=0.7, ax=ax )
```

```
df_height[df_height['Gender'] == 2]['Height'].plot (
    label='Male', kind='hist', alpha=0.7, ax=ax )
```

```
ax.legend()
```

```
ax.set_title('Height Distribution')
```

```
ax.set_xlabel('Height (in centimeters)')
```

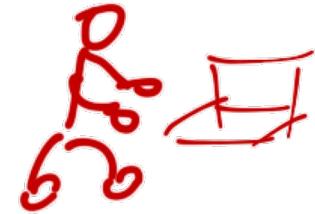


kind='hist' → histogramme

alpha=0.7 → transparence

Label='...' → étiquette

Hands On !



- Exercice : prévoir la taille de quelqu'un à partir du genre
 - Maintenant qu'on a les données, on peut les diviser en deux : **training** et **test**
 - Puis, on identifie les **features** ('Gender') et le **target** ('Height')

```
from sklearn.model_selection import train_test_split

df_train, df_test = train_test_split( df_height, test_size=0.3 )

print ('Train:', df_train.shape, 'Test:', df_test.shape)
```

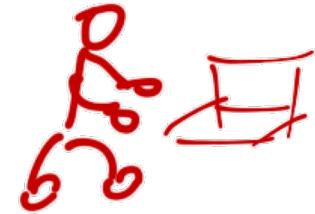
	Gender	Height
417	2	183.757920
731	2	188.714886
623	2	200.219834
746	2	195.184131
783	2	183.331345

Train: (560, 2) Test: (240, 2)

```
x_train = df_train [ ['Gender'] ]
y_train = df_train [ ['Height'] ]
```



Hands On !



- **Exercice : prévoir la taille de quelqu'un à partir du genre**
 - Le jeu de données étant organisé, on peut créer (**DecisionTreeRegressor**) et entraîner (**fit**) le modèle
 - Créez **deux arbres de régression**, un avec le critère '**mae**' et l'autre avec le critère '**mse**'
 - Affichez les deux arbres de régression que vous venez de créer

```
from sklearn.tree import DecisionTreeRegressor, export_text
```

```
rgmae = DecisionTreeRegressor(criterion='mae')
```

```
rgmae.fit(x_train, y_train)
```

```
print ( export_text(rgmae) )
```

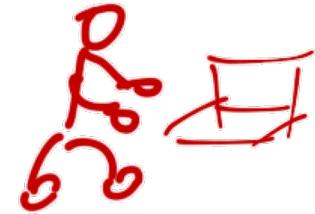
Pensez à la version...
criterion='mae'
 ou
criterion='absolute_error'



Tree MAE

```
|--- feature_0 <= 1.50
|   |--- value: [157.79]
|--- feature_0 >  1.50
|   |--- value: [178.23]
```

Hands On !



- **Exercice : prévoir la taille de quelqu'un à partir du genre**
 - On peut également afficher les deux arbres de manière graphique

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
%matplotlib inline
fig, axs = plt.subplots(1, 2, figsize=(12, 8))
plot_tree(rgmae, feature_names=['Gender'], fontsize=8, ax=axs[0])
plot_tree(rgmse, feature_names=['Gender'], fontsize=8, ax=axs[1])
```

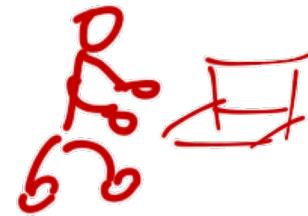
*On n'oublie pas les **imports**...*

*... ni le « **inline** » pour afficher correctement le graphique dans le notebook.*

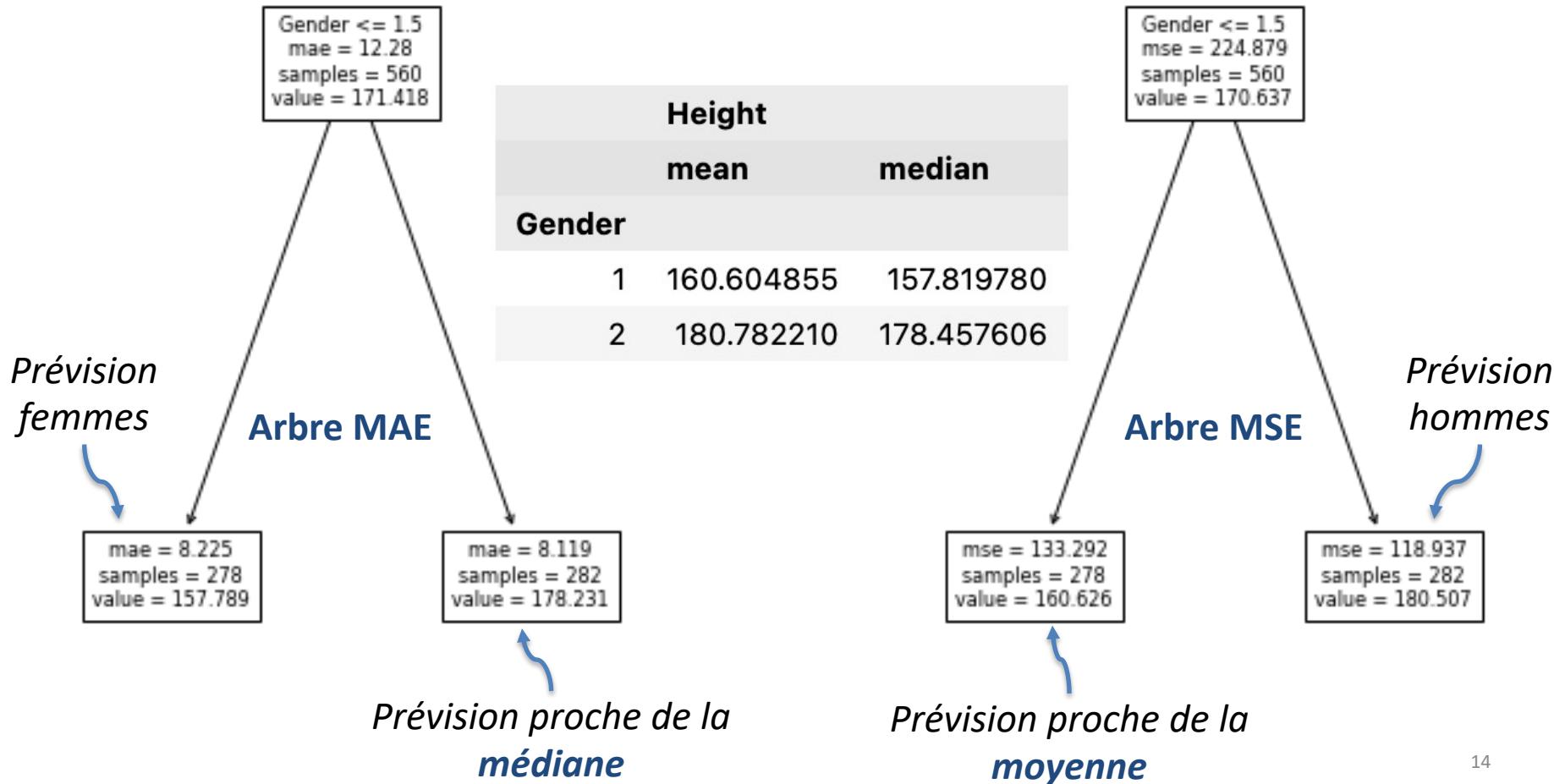
Création d'une figure avec de l'espace pour deux graphiques (1,2 → 1 ligne, 2 colonnes)

*On « **plotte** » chaque arbre dans un de ces espaces (identifiés par la variable **axs**)*

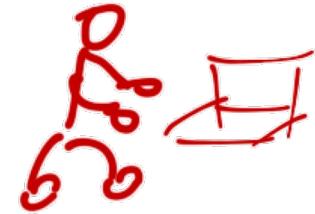
Hands On !



- On analyse un peu les arbres obtenues...



Hands On !



- **Exercice : prévoir la taille de quelqu'un à partir du genre**
 - Evaluer les arbres de régression créées avec les mesures MAE et MSE.
 - Évaluation pour l'arbre créé avec MAE :

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt

y_pred_mae = rgmae.predict( df_test[['Gender']] )

mse_Tmae = mean_squared_error ( df_test['Height'], y_pred_mae )
mae_Tmae = mean_absolute_error ( df_test['Height'], y_pred_mae )
rmse_Tmae = sqrt(mse_Tmae)
```

- Comparaison pour les 2 arbres/les 2 mesures (**print**) :

Arbre	critère	MAE	MSE
	MSE :	135.202	<u>126.672</u>
	MAE :	<u>8.515</u>	8.721
	RMSE :	11.628	<u>11.255</u>

Mesures_error = + c'est petit, mieux c'est



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Régression Linéaire

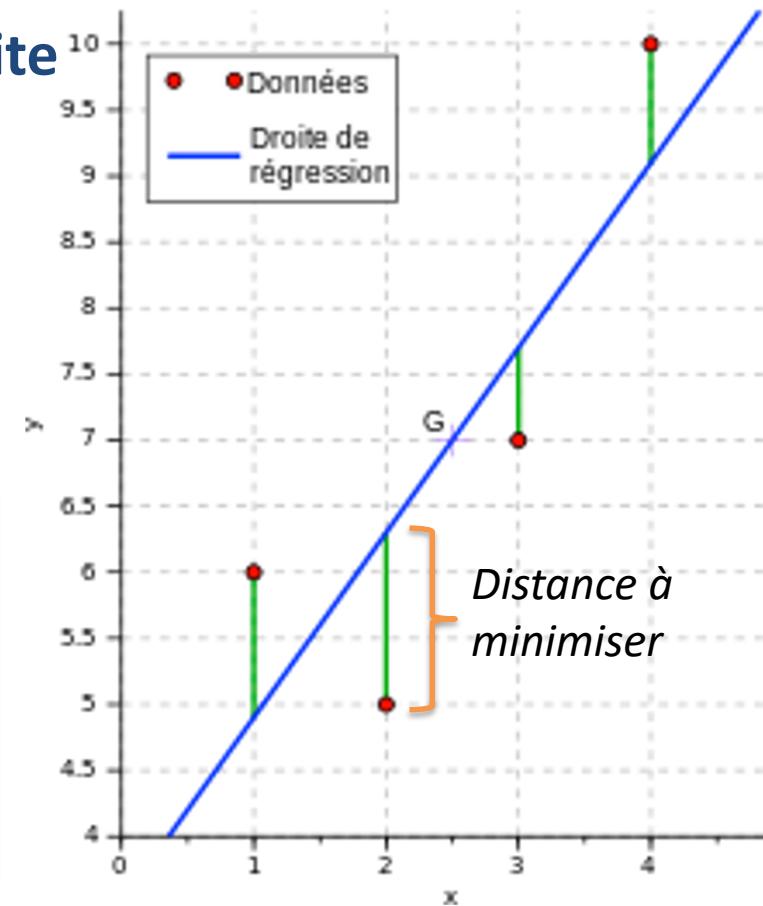
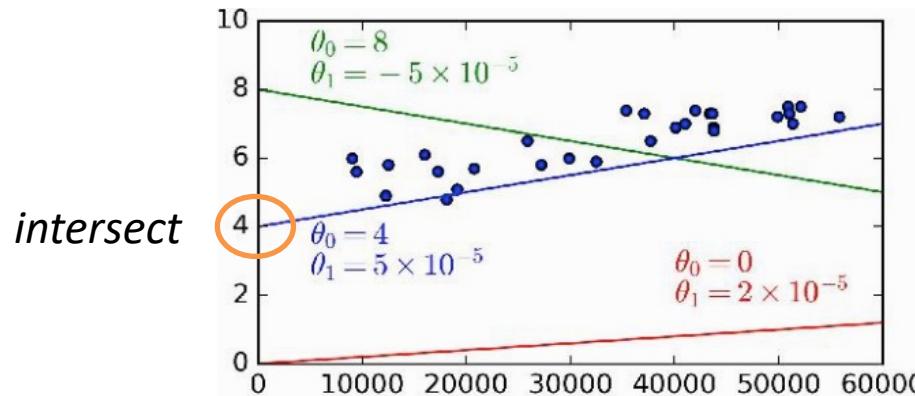
- Bases

- On suppose une relation linéaire entre les données

$$f(x) = a_1x_1 + \cdots + a_px_p + b$$

- Le modèle essaie de retrouver la **droite** qui se rapproche le plus des points

- Minimise **distance** (**erreur**) entre la **prédiction** et la **donnée**
- a_i = **coefficients**
- b = **intersect** (**ordonnée à l'origine**)





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Régression Linéaire

```
from sklearn.linear_model import LinearRegression
```

Création objet modèle

```
model = LinearRegression()
```

On n'oublie pas l'import

```
model.fit(x_train, y_train)
```

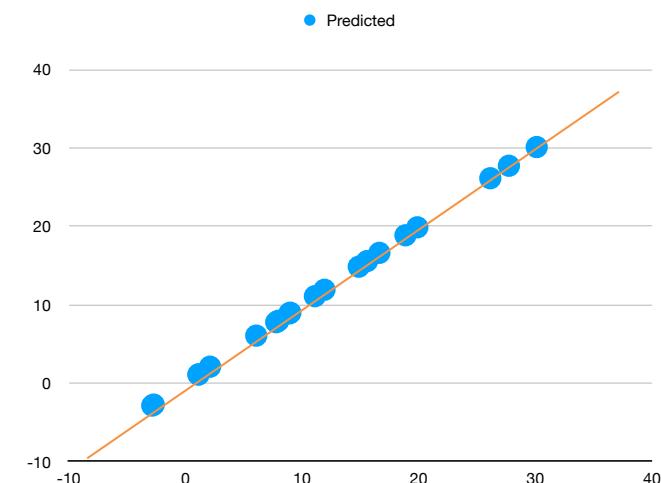
On entraîne le modèle (*fit*)

```
y_pred = model.predict(x_test)
```

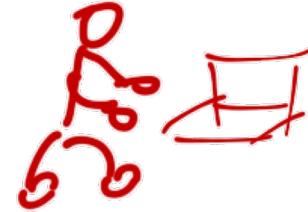
On utilise le modèle pour faire les prédictions (*predict*)

```
intercept = model.intercept_  
coeffs = model.coef_
```

On peut récupérer les valeurs *d'intercept* et le *coefficient* pour *chaque feature*



Hands On !



- **Exercice : prévoir la température à partir de l'humidité**
 - On va commencer par une régression simple à partir d'une seule **feature**
 - Créer un **nouveau Notebook** et charger le fichier « **weather.csv** » à partir de <http://www.kirschpm.fr/cours/PythonDataScience/files/weather.csv> dans un **DataFrame** (**read_csv**)
 - Afficher les informations (**info**) et les premières lignes (**head**)

```
import pandas as pd

df_weather =
pd.read_csv('http://www.kirschpm.fr/cours/
PythonDataScience/files/weather.csv')

df_weather.info()
```

RangeIndex: 10000 entries, 0 to 9999
 Data columns (total 8 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	10000 non-null	object

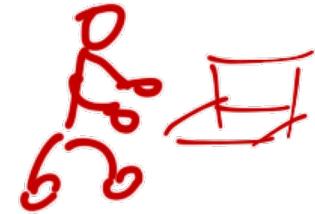
 dtypes: float64(5), int64(2), object(1)
 memory usage: 625.1+ KB

df_weather.head()

	Temperature_c	Humidity	Wind_Speed_kmh	Wind_Bearing_degrees	Visibility_km	Pressure_millibars	Rain
0	-0.555556	0.92	11.2700	130	8.0500	1021.60	0
1	21.111111	0.73	20.9300	330	16.1000	1017.00	1
2	16.600000	0.97	5.9731	193	14.9086	1013.99	1
3	1.600000	0.82	3.2200	300	16.1000	1031.59	1
4	2.194444	0.60	10.8836	116	9.9820	1020.88	1



Hands On !



- **Exercice : prévoir la température à partir de l'humidité**
 - Séparer les données en **training** et **test**
 - Afficher la forme (attribut **shape**) des deux ensembles créés (nb lignes et col.)
 - Afficher les premières lignes du **training** set

On n'oublie pas l'**import**

```
from sklearn.model_selection import train_test_split
```

```
df_train, df_test = train_test_split (df_weather, test_size=0.33,  

                                         random_state=42 )
```

On reçoit deux variables
train et *test*

(6700, 8) (3300, 8)

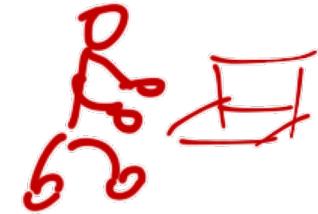
Si on veut pouvoir
 reproduire la séparation

```
print ( df_train.shape, df_test.shape )
```

df_train.head(5)

	Temperature_c	Humidity	Wind_Speed_kmh	Wind_Bearing_degrees	Visibility
8371	11.044444	0.67	15.1984		149 9.98
5027	23.961111	0.47	8.3076		142 9.98
9234	-1.250000	0.92	10.9641		202 4.00
3944	12.222222	0.47	20.3021		271 10.25

Hands On !



- **Exercice : prévoir la température à partir de l'humidité**
 - On va séparer maintenant les **features** (humidité) et le **target** (température)

```
feature_names = ['Humidity']
target = ['Temperature_c']
```

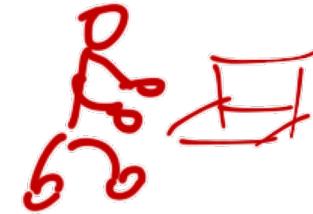
- Séparer un DataFrame **x_train** avec la colonne « **Humidity** » (**feature**) et un **y_train** avec la colonne « **Temperature_c** » (**target**)
- Faire la **même chose** pour les données de test (**x_test** et **y_test**)

```
x_train = df_train [ feature_names ]
y_train = df_train [ target ]
```

```
x_train.describe()
```

	Humidity
count	6700.000000
mean	0.734115
std	0.196582
min	0.000000
25%	0.600000
50%	0.780000
75%	0.900000
max	1.000000

Hands On !



- Exercice : prévoir la température à partir de l'humidité**

- Maintenant qu'on a séparé nos données, on peut créer un nouveau modèle de Régression Linéaire et l'entraîner.

```
from sklearn.linear_model import LinearRegression
```

On n'oublie pas l'import

```
model = LinearRegression()
```

Création de l'objet modèle
Entrainement (**fit**) du modèle

```
model.fit(x_train, y_train)
```

- On peut également afficher les valeurs de l'**intercept** et du **coefficient**

```
intercept = model.intercept_
coeffs = model.coef_
```

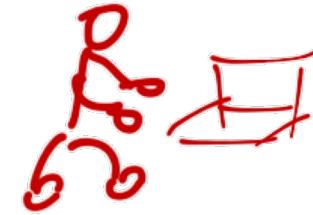
*On peut récupérer les attributs **intercept_** et **coef_** afin de restituer l'équation*

```
print ('intercept :', intercept, 'Coeff humidité :', coeffs)
```

```
print('Temperature = ', intercept, '+', coeffs[0], 'x Humidity')
```

Temperature = [34.57484017] + [-30.88864271] x Humidity

Hands On !



- **Exercice : prévoir la température à partir de l'humidité**
 - Une fois entraîné, on peut tester le modèle avec les données de test (`x_test`)

```
y_pred = model.predict ( x_test )
```

- On peut aussi essayer de **visualiser** les données obtenues (**prédites**) et leur **corrélation** avec celles **attendues**

On choisit les informations à afficher (données attendues, prédites et l'humidité)

```
visu = pnd.DataFrame(y_test[target])
visu['Predicted'] = y_pred
visu['Humidity'] = x_test[feature_names]
```

On prépare un DataFrame pour la visualisation

```
visu.rename(columns={'Temperature_c':'Expected'},
            inplace=True)
```

```
visu.head()
```

On change le nom de la colonne « Temperature_c » en « Expected » (données attendues)

	Expected	Predicted	Humidity
6252	-2.727778	18.821632	0.51
4684	11.094444	8.010607	0.86
1731	1.122222	7.392835	0.88
4742	-2.850000	14.806109	0.64
4521	7.777778	4.921743	0.96

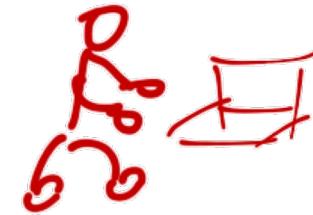


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Hands On !



On utilise la bibliothèque `matplotlib.pyplot`, donc `import`

```
import matplotlib.pyplot as plt  
%matplotlib inline
```

Nécessaire pour afficher correctement les graphiques sur le notebook

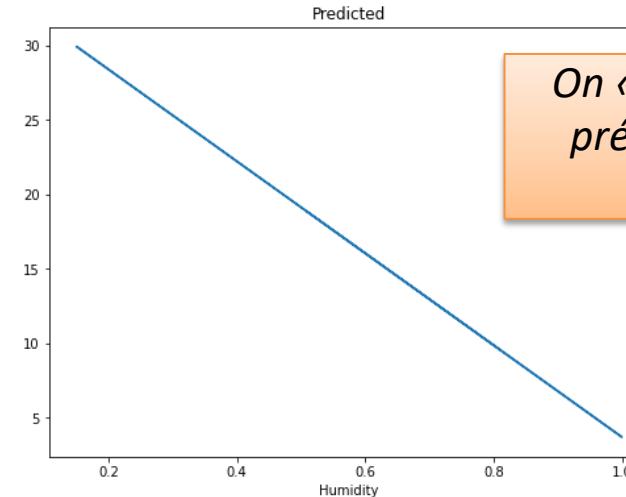
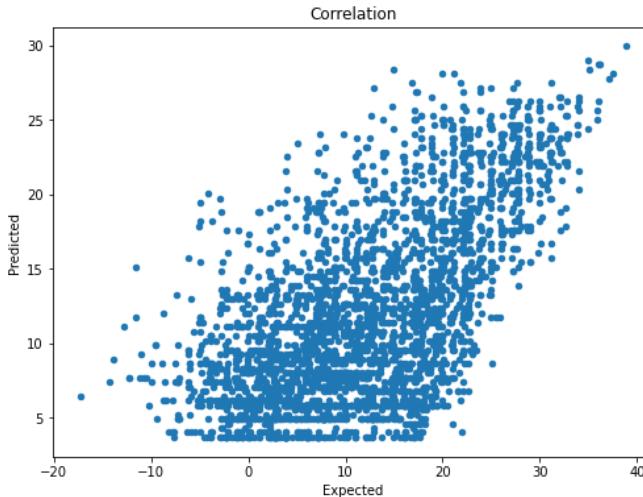
```
fig, axs = plt.subplots(1, 2, figsize=(18, 6))
```

Création d'une image avec la place pour 2 graphiques

```
visu.plot.scatter ( x='Expected', y='Predicted',  
                    title='Correlation', ax=axs[0] )
```

On « plotte » la corrélation entre attendu et prédit

```
visu.set_index('Humidity')[['Predicted']].plot ( title='Predicted',  
                                                kind='line', ax=axs[1] )
```



On « plotte » les valeurs prédites par rapport à l'humidité

Temperature = [34.57] + [-30.89] x Humidity

Mesures d'évaluation

- Différences **mesures d'évaluation** possibles
 - MAE, MSE et RMSE (comme pour l'arbre de régression)

R2 Score
(coefficient de détermination)

Mesure l'**adéquation** entre les prévisions et les données **réelles**

À quel point la prévision est « mieux » que la **moyenne**

$$R^2 = 1 - \frac{\sum(Y_{réel} - Y_{prédict})^2}{\sum(Y_{réel} - Y_{moy})^2}$$

Explained Variance

Mesure la **dispersion** des données **prédites** par rapport aux données **réelles**

Scores entre [0, 1]

- 1 → prédiction parfaite
- 0 → modèle trop pauvre

$$\text{Explained Variance} = 1 - \frac{\text{Var}\{Y_{réel} - Y_{prédict}\}}{\text{Var}\{Y_{réel}\}}$$



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Mesures d'évaluation

```
from sklearn.metrics import mean_absolute_error,  
                           mean_squared_error,  
                           r2_score,  
                           explained_variance_score  
  
from math import sqrt
```

Les métriques se trouvent dans la bibliothèque **sklearn.metrics**.
Il faut penser à l'**import**.

```
mae = mean_absolute_error ( y_test , y_pred )
```

```
mse = mean_squared_error ( y_test , y_pred )  
rmse = sqrt(mse)
```

```
r2 = r2_score ( y_test , y_pred )
```

```
expvar = explained_variance_score ( y_test , y_pred )
```

Quelle que soit la métrique, on va toujours comparer les **données réelles** aux **données prédictives** par le modèle

Hands On !

- **Exercice : évaluer notre modèle de température**

- Évaluer le modèle avec les mesures MSA, MSE et RMSE, puis avec la variance.
- Afficher les résultats obtenus par ces métriques.

Suggestion d'affichage : texte formaté

```
print ('texte à trous : un {0} chiffre {1:d} décimal {2:.2f}'.format (
    'trou',
    234,
    2.346) )
```

On laisse des **trous à remplir** ({}) avec des indications sur le **format** ({ :d } , { :.2f }), qu'on remplit avec l'opération **format**

```
print ('MAE={:.3f}'.format(mae) )
```

```
print ('RMSE={0:.3f} \t MSE={1:.3f} '.format(rmse, mse) )
```

```
print ('Explained Variance={:.3f}'.format(expvar) )
```

MAE=6.006
RMSE=7.319 MSE=53.564
Explained Variance=0.385



Régression Linéaire

- **Modèle régression linéaire multiple**
 - Utiliser **plusieurs variables (features)** peut aider à améliorer la qualité du modèle

- **Exemple : données météo**

- Utiliser plus que seule l'humidité peut contribuer à avoir un modèle de meilleure qualité

Problème : données non numériques



Temperature_c	Humidity	Wind_Speed_kmh	Wind_Bearing_degrees	Visibility_km	Pressure_millibars	Rain	Description
11.044444	0.67	15.1984	149	9.9820	1026.72	1	Normal
23.961111	0.47	8.3076	142	9.9820	1018.97	1	Warm
-1.250000	0.92	10.9641	202	4.0089	1032.08	0	Cold
12.222222	0.47	20.3021	271	10.2557	1014.61	1	Normal
-1.227778	1.00	7.8890	102	2.9624	1019.31	0	Cold



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Encoding

- **Encoding**

- Transformer les données « texte » en données numériques
- La bibliothèque **Pandas** propose un *encoding* de type **One Hot** appelé « `get_dummies` »

```
df_weather.value_counts( df_weather['Description'] )
```

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

Description		
Normal	4992	
Warm	2507	
Cold	2501	

dtype: int64

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



Hands On !

- Exercice : transformer les données de la colonne « Description » en données numériques**

- Afficher le nombre de valeurs présentes dans la colonne « Description »

```
df_weather.value_counts( df_weather['Description'] )
```

- Utiliser l'opération « `get_dummies` » pour convertir ces valeurs en chiffres

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

Description		
Normal	4992	
Warm	2507	
Cold	2501	
		dtype: int64

On obtient 1 colonne en moins

Data columns (total 9 columns):	Column	Non-Null Count	Dtype
#	Column	-----	-----
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

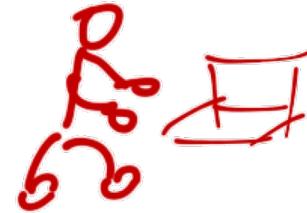
Permet de représenter la 1^{ère} catégorie par la combinaison de 0 sur les autres

	Description_Normal	Description_Warm
	0	1
0	0	1
1	1	0
0	0	0
0	0	0

`df_dummies.info()`

`df_dummies.head()`

Hands On !



- **Exercice : réaliser un nouveau modèle de régression linéaire**
 - On utiliser maintenant toutes les variables du dataset « `df_dummies` »
 - Refaire la séparation **training** et **test**

```
df_train, df_test = train_test_split ( df_dummies, test_size=0.33,
                                         random_state=42)
```

- Pour les Y set (`y_train` et `y_test`), utiliser la colonne « `Temperature_c` »
- Pour les X set (`x_train` et `x_test`), **supprimer** la colonne « `Temperature_c` »

```
y_train = df_train[ [ 'Temperature_c' ] ]
x_train = df_train.drop(['Temperature_c'], axis='columns')
x_train.head()
```

	Humidity	Wind_Speed_kmh	Wind_Bearing_degrees	Visibility_km	Pressure_millibars	Rain	Description_Normal	Description_Warm
8371	0.67	15.1984		149	9.9820	1026.72	1	1
5027	0.47	8.3076		142	9.9820	1018.97	1	0
9234	0.92	10.9641		202	4.0089	1032.08	0	0
3944	0.47	20.3021		271	10.2557	1014.61	1	0
2662	1.00	7.0000		102	9.0001	1010.81	0	0

Hands On !

- Exercice : réaliser un nouveau modèle de régression linéaire
 - Créer un **nouveau modèle** et l'entrainer avec les données de training
 - Afficher l'**intercept** et les **coefficients** obtenus

```
model_full = LinearRegression()  
  
model_full.fit ( x_train , y_train )  
  
print ( 'Intercept :', model_full.intercept_ )  
print ( 'Coefficients :', model_full.coef_ )
```

Intercept : 3.6449405327633073
Coefficients : [-8.12735036e+00 -7.16582510e-02 1.13431836e-03 5.47996684e-02
7.55608339e-05 5.67566186e+00 8.48376238e+00 1.91937072e+01]

Humidity : -8.127350361370153
Wind_Speed_kmh : -0.07165825096198843
Wind_Bearing_degrees : 0.001134318363873788
Visibility_km : 0.05479966838317961
Pressure_millibars : 7.556083387600545e-05
Rain : 5.675661864415515

Suggestion d'affichage

```
liste_coeffs = model_full.coef_.ravel()  
for i,coef in enumerate (liste_coeffs) :  
    print (x_train.columns[i], ":", coef)
```

Hands On !

- Exercice : réaliser un nouveau modèle de régression linéaire
 - Tester le modèle avec les données de test

```
y_pred_full = model_full.predict ( x_test )
```

- Evaluer le modèle obtenu avec différentes métriques (MSA, MSE et RMSE, et *explained variance*)

```
mae_full = mean_absolute_error ( y_test, y_pred_full )
mse_full = mean_squared_error ( y_test, y_pred_full )
rmse_full = sqrt(mse_full)
var_full = explained_variance_score( y_test, y_pred_full)
```

- Afficher les valeurs obtenues

MAE=6.006
RMSE=7.319
Explained Variance=0.385

Valeurs modèle **régression simple**
MSE=53.564

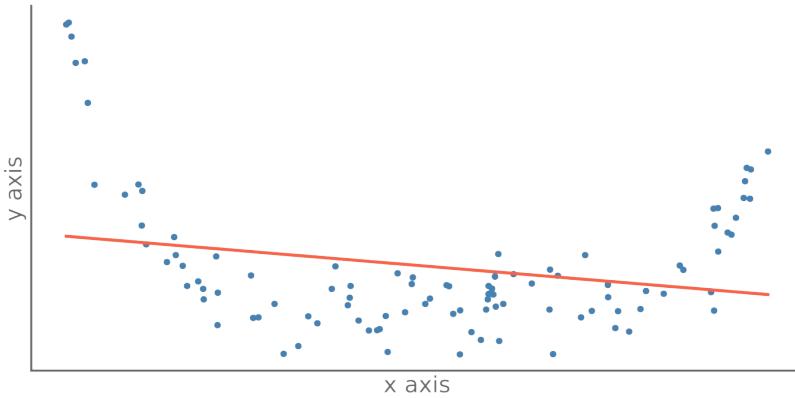
Valeurs **nouveau modèle (régression multiple)**

MAE = 2.848
RMSE = 3.491
MSE = 12.187
Explained Variance = 0.860

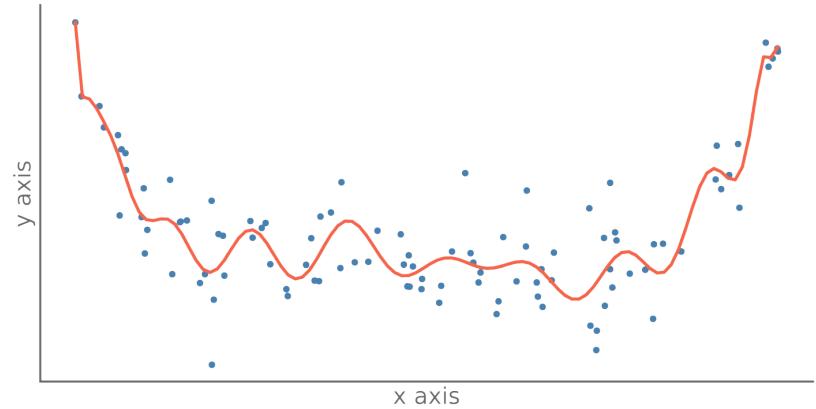
Plus explained variance est proche de 1 et meilleur est le modèle

Risques d'un « mauvais » modèle

- Risques de « sous-apprentissage » (*underfitting*) et de « sûr-apprentissage » (*overfitting*)



Underfitting



Overfitting