2022/2023

Régression Linéaire

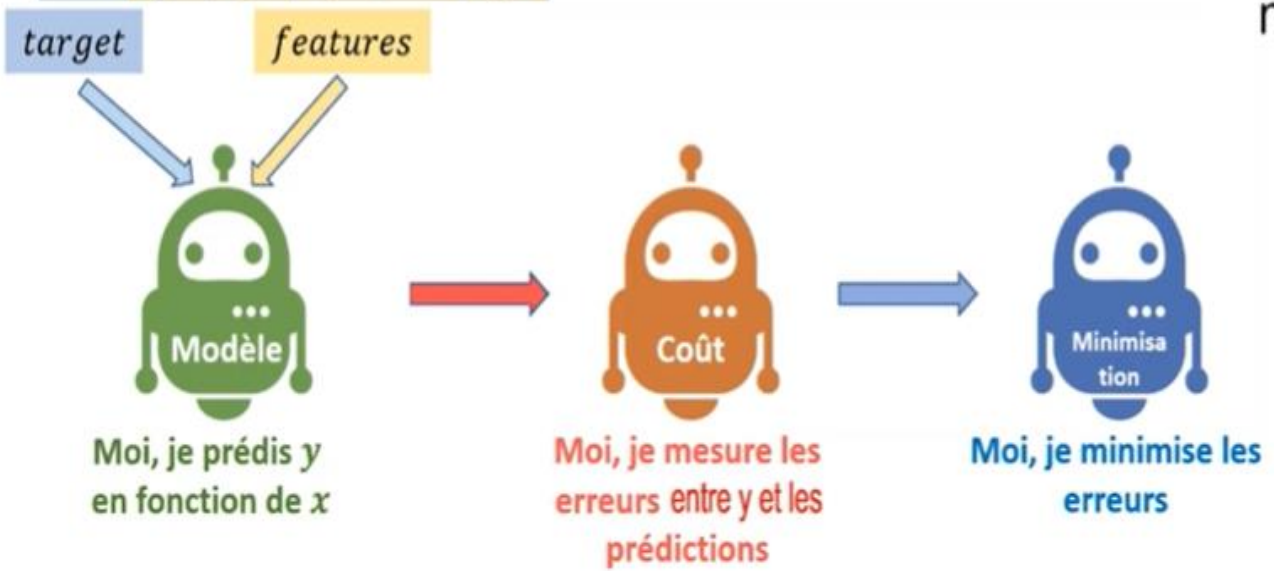
Réalisé par: Naïma Daghfous

Schéma de l'Apprentissage Supervisé

$Dataset\ (x, y)$

y	x_1	x_2	x_3	...	x_n
$y^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	...	$x_n^{(1)}$
$y^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	...	$x_n^{(2)}$
$y^{(3)}$	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	...	$x_n^{(3)}$
...	...	...	...	...	...
$y^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	$x_3^{(m)}$	...	$x_n^{(m)}$

1. Dataset
   y : Target
   x : features
2. Modèle
   paramètres
3. Fonction Coût
4. Algorithme de minimisation



Régression & Classification

L'apprentissage supervisé

Je sais ce que je cherche à prédire et j'ai déjà des données en guise d'exemple à fournir à ma machine

SUPERVISÉ

La régression

Je cherche à prédire une valeur numérique, un chiffre, un nombre ?

Exemples : ventes, marge, stock, température, pression, taille, poids, quantité…

La classification

Je cherche à prédire une catégorie, une dimension, une classe ?

Exemples : oui / non, vrai / faux, homme / femme, segments, espèces d'animaux…

Régression & Classification

- note par X l'espace des variables d'entrées (ex. $\mathbb{R}^D$)
- On dénote par Y l'espace des variables de sortie (ex. $\mathbb{R}$)
-
- Ayant un ensemble de données $D \subset X \times Y$, trouver une fonction :

$$f: X \rightarrow Y$$

Les problèmes sont classés par type de domaine de sortie:

- Si Y= $\mathbb{R}$, on parle alors de régression.
- Si Y est un ensemble discret fini, on parle de classification.
- Si Y a 2 éléments, on parle de classification binaire.

Régression Linéaire

La variable cible (**Y**) est quantitative

Le prix d'une maison

La variable (**X**) est quantitative ou qualitative

Nombre de pièces
Localisation
Avec ou sans balcon
Nombre de façades
…

Objectif : de trouver une fonction dite de prédiction ou une fonction coût qui décrit la relation entre ***X*** et ***Y***

La fonction recherchée est : **Y= f(X)** Avec **f(X)** une fonction linéaire.

Régression Linéaire

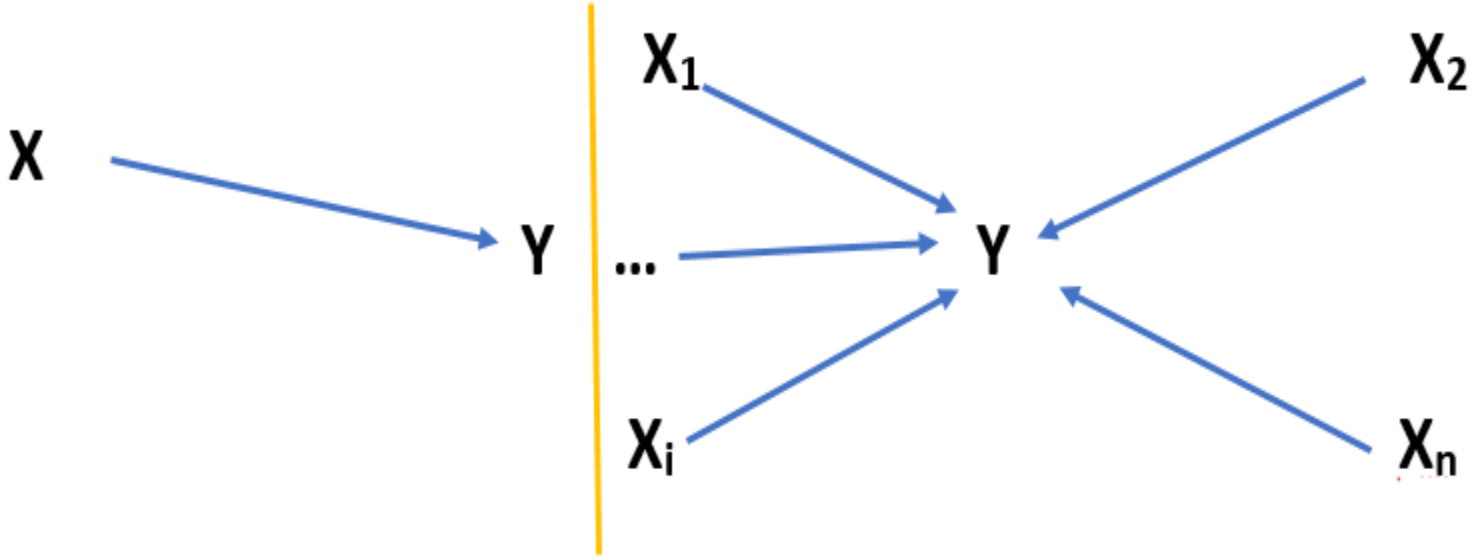
- La régression consiste à approcher une variable à partir d'un ou plusieurs variables qui lui sont corrélées

Une seule variable explicative X

Régression simple

Plusieurs variables explicatives $X_j (j=1,\ldots,n)$

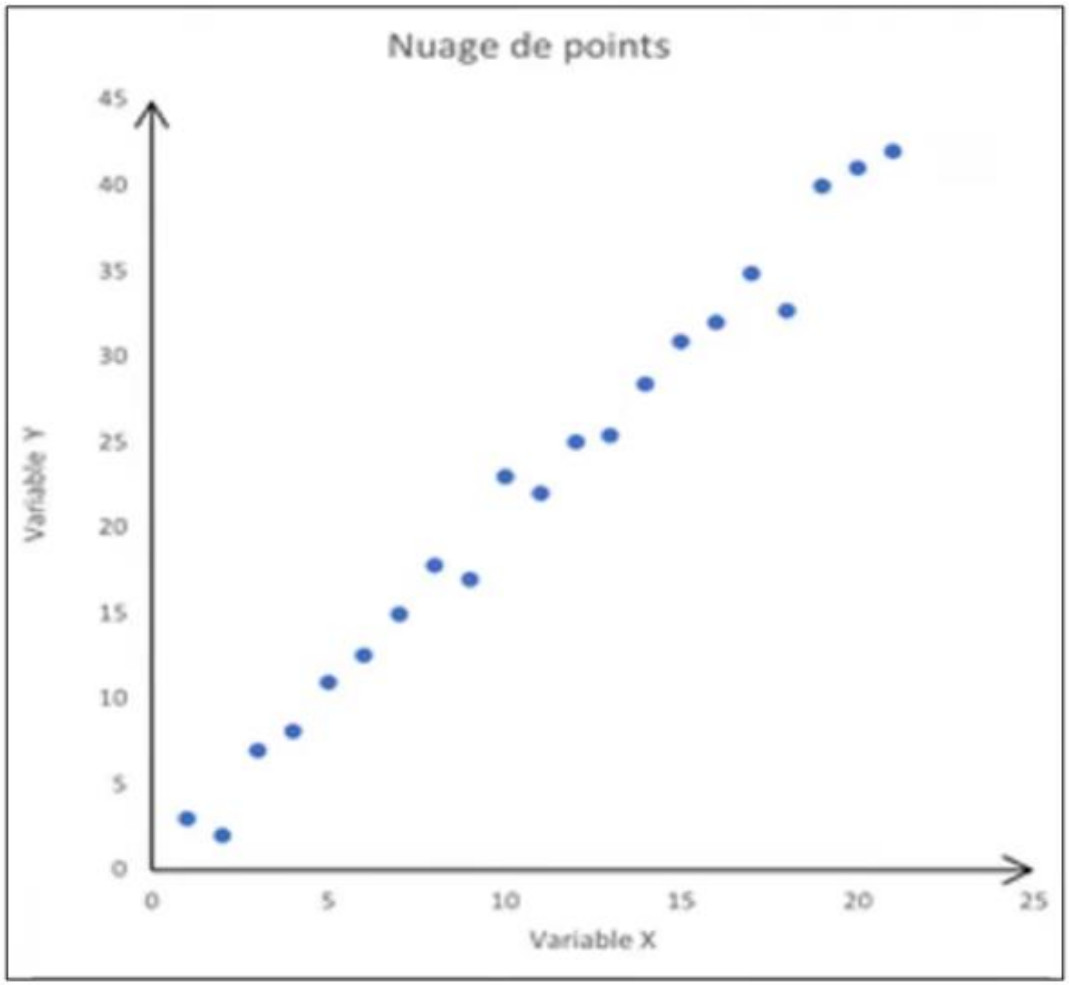
Régression multiple



$Y = aX + b + \varepsilon$ 1 variable explicative

$Y = a_1X_1 + a_2X_2 + a_3X_3 + b + \varepsilon$ 3 variables explicatives indépendantes

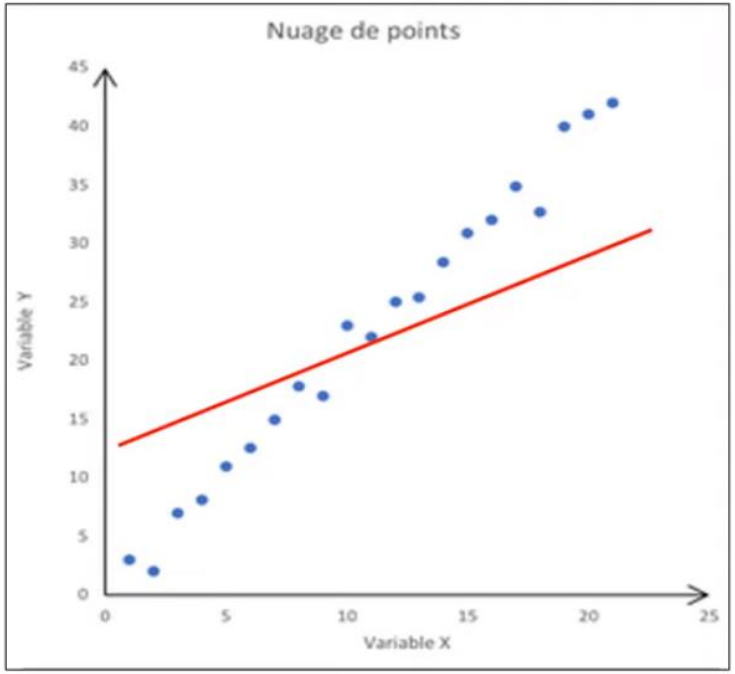
RL DataSet



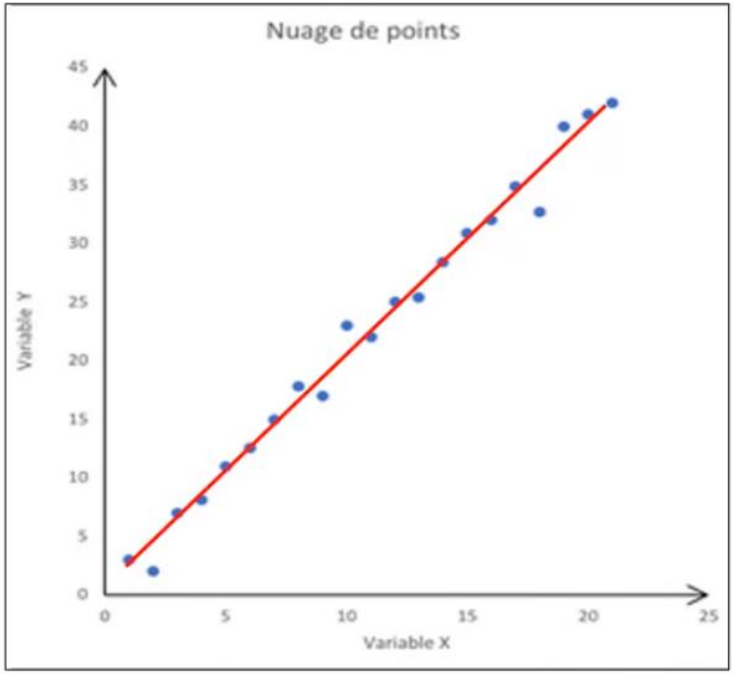
RL Model

$$y = a.x + b$$

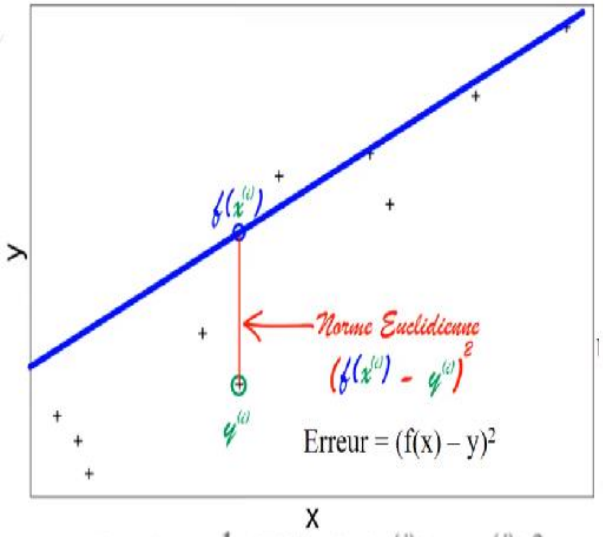
Modèle initial (paramètres aléatoires)



Modèle final (paramètres trouvés)



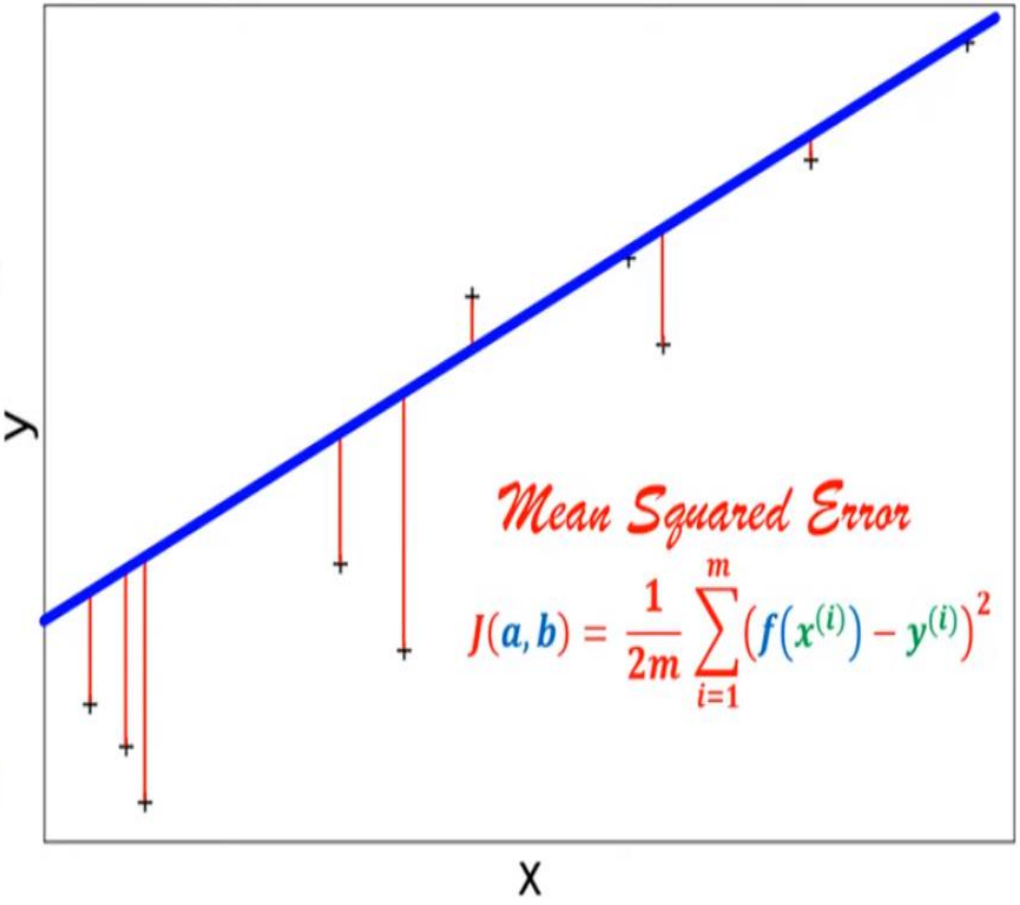
RL Fonction Cout



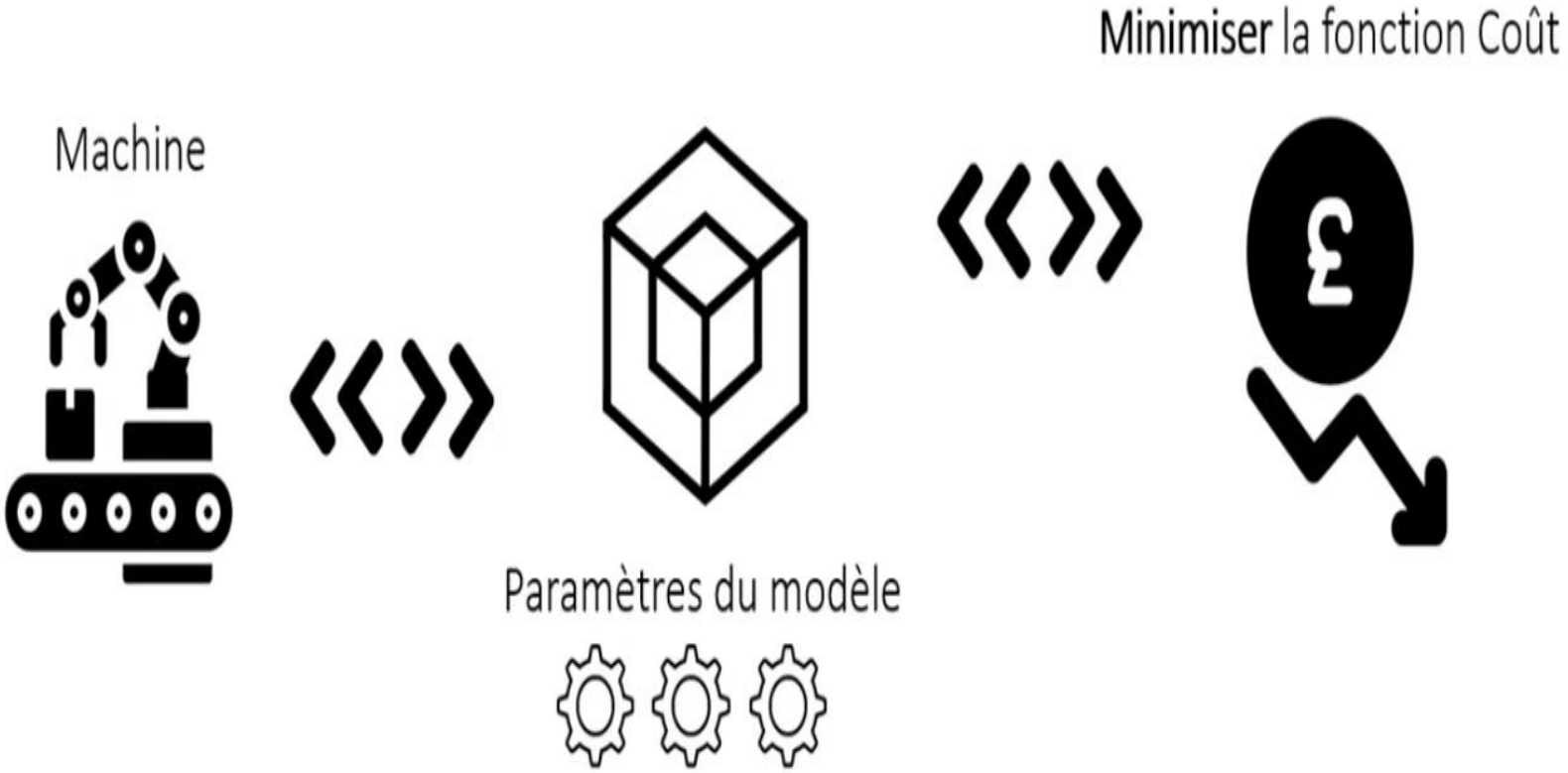
$$J(a,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f(x^{(i)}) - y^{(i)}\right)^2$$

$$J(a,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(ax^{(i)} + b - y^{(i)}\right)^2$$

Erreur Quadratique Moyenne



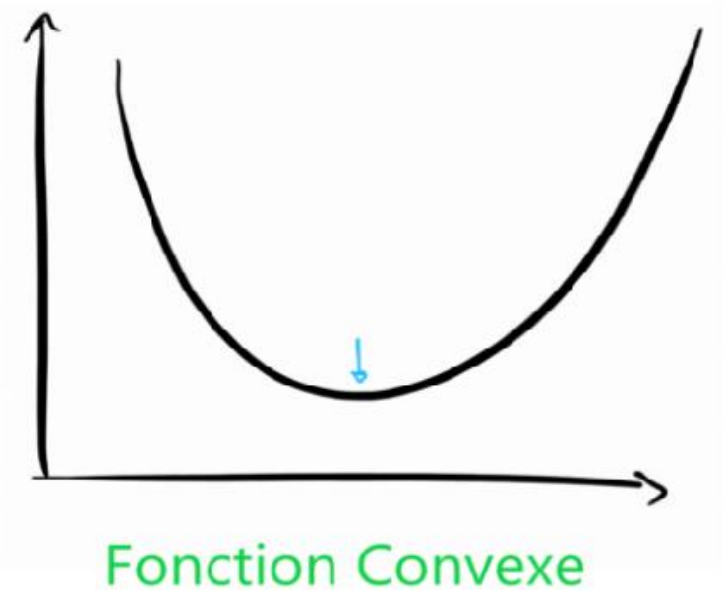
RL Fonction Cout



- ✓ **Méthode des moindres carrés**
- ✓ **Gradient Descent**

RL & Gradient Descent

- La **Descente de Gradient**, (ou ***Gradient Descent***) est un des algorithmes les plus importants d'optimisation, puissant qui permet d'entraîner les modèles de **régression linéaire**
Il permet de trouver le **minimum** de n'importe quelle fonction **convexe** en convergeant progressivement vers celui-

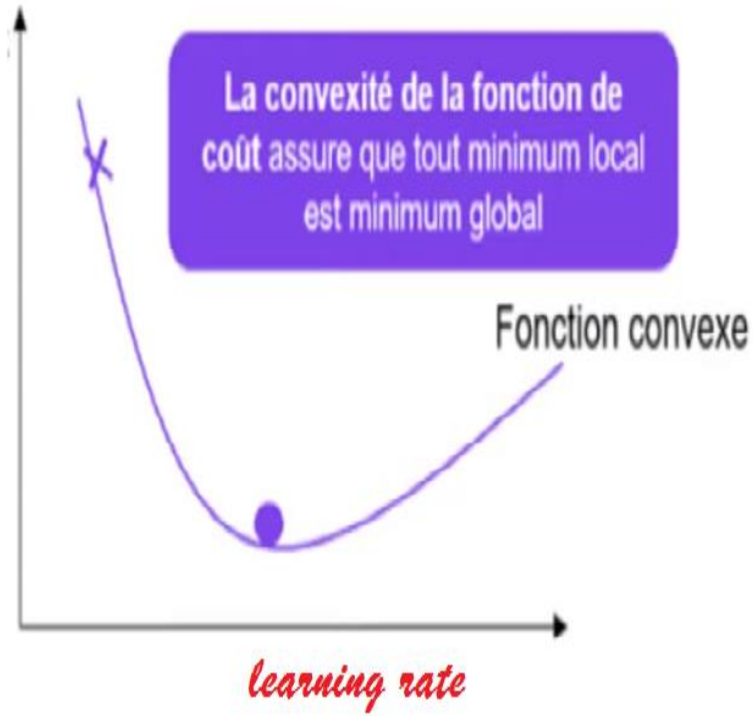


RL & Gradient Descent

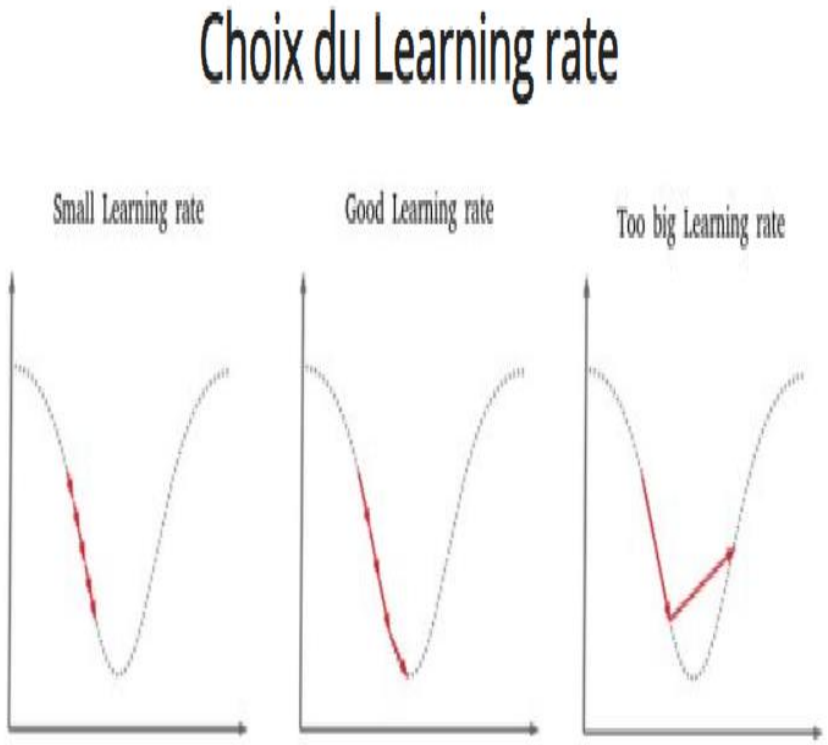
- Principe de la **Descente de Gradient :** Perdu en pleine montagne, votre but est de rejoindre un refuge situé au point le plus bas de la vallée. 2 étapes à faire
  - Depuis votre position actuelle, vous cherchez tout autour de vous la direction de là où la pente descend **le plus fort**.
  - Une fois que vous avez trouvé cette direction, vous la suivez sur une **certaine distance α** (disons que vous marchez 300 mètres) puis vous répétez l'opération de l'étape 1.

=> cette stratégie est l'algorithme de la Descente de Gradient, on appelle la distance **α** ***Learning Rate***, que l'on pourrait traduire par *vitesse d'apprentissage.*

RL & Gradient Descent



$$\begin{pmatrix} a_i \\ b_i \end{pmatrix} := \begin{pmatrix} a_i \\ b_i \end{pmatrix} - \alpha \frac{\partial}{\partial \beta_i} C(a, b)$$



RL Coder un Exemple

X : Surface

Y : Loyer

```
surface;loyer
37;1330
32;1400
26;904
30;955
70;2545
24;900
41;1560
67;1960
63;2000
75;2600
81;3280
347;16000
26;980
36;1250
19;752
20;815
25;1147
35;1500
120;3587
40;1355
30;1245
25;1100
21;1120
44;1225
...
```

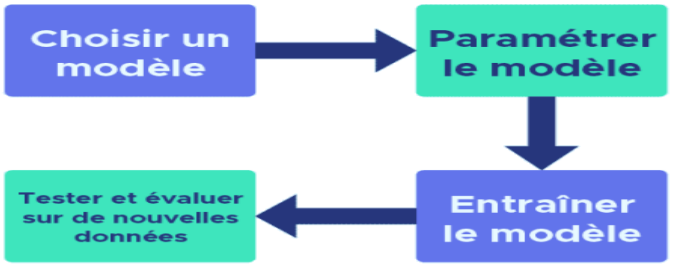
RL Coder un Exemple

- **Pandas** est une librairie python qui permet de manipuler facilement des données à analyser :
  - manipuler des tableaux de données avec des étiquettes de variables (colonnes) et d'individus (lignes). Ces tableaux sont appelés DataFrames.
  - on peut facilement lire et écrire ces dataframes à partir ou vers un fichier tabulé.
  - on peut faciler tracer des graphes à partir de ces DataFrames grâce à matplotlib.

- **Matplotlib** : c'est une librairie qui permet de tracer des graphes (dans le sens graphiques)

RL Coder un Exemple

- **Scikit-learn** est une librairie Python open source qui implémente une gamme d'algorithmes d'apprentissage automatique, de prétraitement, de validation…
- Les étapes de modélisation sont souvent les mêmes :



- **Choisir un modèle** en important la classe appropriée de Scikit-Learn.
- **Paramétrer le modèle**. Si vous êtes déjà sûrs des paramètres que vous voulez utiliser vous pouvez les renseigner à la main
- **Entraîner le modèle** sur le jeu d'apprentissage à l'aide de la méthode fit.
- **Tester le modèle** sur de nouvelles données :
  - Dans le cadre d'apprentissage supervisé, utiliser la méthode **predict** sur les données test.
  - Dans le cadre d'apprentissage non supervisé, nous utilisons les méthodes **transform** ou **predict**.
- Ces 4 étapes sont en général communes à l'utilisation d'un grand nombre de modèles disponibles dans la librairie

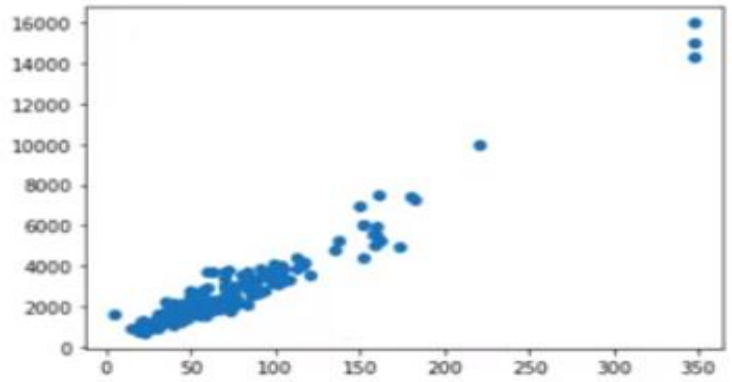
RL En Pratique : DataSet

```
# importer les librairies
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
# importer dataset
Data=pd.read_csv('LoyersMaisons.csv',sep=";")
```

```
# afficher nuage de points
plt.scatter(Data['surface'],Data['loyer'])
```

```
<matplotlib.collections.PathCollection at 0x1b6eac4c790>
```




```
# exclure les données dont le loyer supérieur à 10000
Data=Data[Data['loyer']<=10000]
```

```
X=Data.iloc[:,:-1].values
Y=Data.iloc[:,-1].values
```

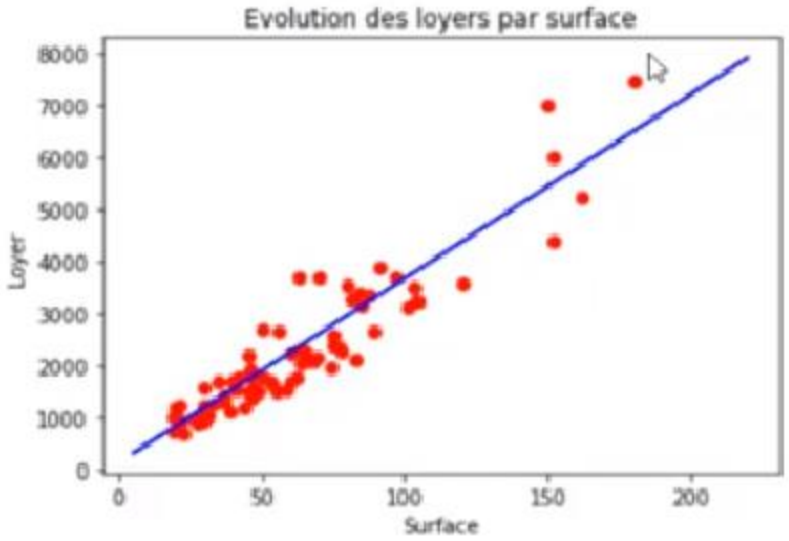
```
# diviser le dataset
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=1.0/3)
```

RL En Pratique : Model

```
# construire le modèle
regressor=LinearRegression()
regressor.fit(X_train,Y_train)
```

```
# faire des prédictions
Y_pred=regressor.predict(X_test)
```

```
#Visualiser les resultats
plt.scatter(X_test,Y_test,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title('Evolution des loyers par surface')
plt.xlabel('Surface')
plt.ylabel('Loyer')
```



RL En Pratique : Model

```
Y_test
```

```
array([ 950, 7000, 1820, 2282, 2650, 3400, 2400,  966, 3234, 4407, 1245,
       1200, 1048, 2100, 3500, 1700, 1800, 3700, 1700, 1200, 1790, 7450,
       1689,  880, 1560, 1958, 1550, 1795,  752, 3350, 1400, 1500,  700,
        955, 1950, 1600, 2250, 1114, 2200, 6000, 1330, 3587, 1225, 3130,
        900, 1750, 2359, 1445, 3700, 1290, 3900, 3550, 2180, 3705, 1500,
       3280, 3150, 1000,  980, 1090, 1600, 1765, 1100, 1850, 1250, 1620,
       2116, 2700, 1700, 2044, 3243, 2600, 1420, 5233, 2324, 2650, 1520,
       2102, 2300], dtype=int64)
```

```
Y_pred
```

```
array([ 892.8238885 , 5444.33873695, 1774.90040952, 2903.95835642,
       3292.07202567, 3115.65672146, 2798.1091739 , 1210.37143607,
       3856.60099912, 5514.90485863, 1210.37143607,  857.54082766,
       1245.65449691, 2445.27856549, 3786.03487744, 1563.20204447,
       1633.76816615, 2621.69386969, 1386.78674027,  857.54082766,
       1916.03265288, 6502.83056217, 2021.8818354 ,  892.8238885 ,
       1598.48510531, 2762.82611306, 2233.58020045, 1916.03265288,
        822.25776682, 3256.78896483, 1810.18347036, 1739.61734868,
        963.39001018, 1210.37143607, 1774.90040952, 1210.37143607,
```

RL Coder un Exemple(2)

X_1: Année, X_2:kilométrage

X_3: nombre de chevaux, X_4:Marque

Y : Prix

	A	B	C	D	E	F
1	Année	Kilométrage	Nb chevaux	Marque	Prix	
2	2005	250000	6	Audi	90000	
3	2004	215000	7	Audi	85000	
4	2010	145000	8	BMW	110000	
5	2015	90000	6	BMW	130000	
6	2017	65000	9	Mercedes	160000	
7	2012	150000	7	BMW	120000	
8	2010	160000	7	Audi	95000	
9	2012	155000	7	Mercedes	120000	
10	2015	85000	6	BMW	132000	
11	2018	52000	6	Mercedes	175000	
12	2014	130000	7	Audi	140000	
13	2013	130000	7	Mercedes	140000	
14	2013	170000	6	BMW	135000	
15	2016	110000	6	Mercedes	155000	
16	2019	55000	7	Audi	210000	
17	2011	140000	7	Mercedes	111000	
18	2009	185000	6	BMW	90000	
19	2008	170000	6	Mercedes	105000	

Feuil1

RL Coder un Exemple(2)

```
#importer les librairies
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
# emplacement du fichier courant
import os
os.getcwd()
```

```
C:\\Users\\MR'
```

```
#importer le dataset
Data=pd.read_excel('VoituresOccasion.xlsx')
X=Data.iloc[:,:-1]
Y=Data.iloc[:,-1]
```

```
X
```

	Année	Kilométrage	Nb chevaux	Marque
0	2005	250000	6	Audi
1	2004	215000	7	Audi
2	2010	145000	8	BMW
3	2015	90000	6	BMW
4	2017	65000	9	Mercedes

RL Coder un Exemple(2)

	Année	Kilométrage	Nb chevaux	Marque
0	2005	250000	6	Audi
1	2004	215000	7	Audi
2	2010	145000	8	BMW
3	2015	90000	6	BMW
4	2017	65000	9	Mercedes
5	2012	150000	7	BMW
6	2010	160000	7	Audi
7	2012	155000	7	Mercedes
8	2015	85000	6	BMW
9	2018	52000	6	Mercedes
10	2014	130000	7	Audi

```
# Gérer les variables catégoriques
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("Marque", OneHotEncoder(), [3])], remainder = 'passthrough')
X = ct.fit_transform(X)
```

```
X
```

```
array([[1.0, 0.0, 0.0, 2005, 250000, 6],
       [1.0, 0.0, 0.0, 2004, 215000, 7],
       [0.0, 1.0, 0.0, 2010, 145000, 8],
       [0.0, 1.0, 0.0, 2015, 90000, 6],
       [0.0, 0.0, 1.0, 2017, 65000, 9],
```

RL Coder un Exemple(2)

```
#Diviser le dataset
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2)
```

```
# construire le modèle
regressor=LinearRegression()
regressor.fit(X_train,Y_train)
```

```
LinearRegression()
```

```
#Faire des prédictions
Y_pred=regressor.predict(X_test)
```

```
Y_pred
```

```
array([112077.94647767, 108350.20705299, 110811.50799141, 176828.9262092 ,
       188022.55472398, 175949.40089421, 156636.18474465, 175849.25747913,
       171083.92306211, 116933.81628657])
```

```
Y_test
```

```
array([120000, 115000, 115000, 190000, 205000, 160000, 130000, 180000,
       190000, 110000], dtype=int64)
```

Annexe

* Accès à une colonne :

- `df['A']` : renvoie la Series correspondant à la colonne de label A :

```
a1    1.1
a2    2.7
a3    5.3
```

* Accès à un sous-ensemble du dataframe avec les numéros des lignes et colonnes :

- `df.iloc[1]` : renvoie la deuxième ligne.
- `df.iloc[1:3,[0, 2]]` : renvoie le dataframe avec les lignes 1 à 3 exclue, et les colonnes numéros 0 et 2.
- `df.iloc[:,2:4]` : renvoie toutes les lignes et les colonnes 2 à 4 exclue.
- `df.iloc[1,2]` : renvoie la valeur à la ligne 2 et la colonne 3.
- `df.iat[1,2]` : renvoie la valeur à la ligne 2 et la colonne 3, mais c'est la façon recommandée d'accéder aux valeurs.
- on peut aussi faire une affectation pour changer la valeur : `df.iat[1, 2] = 7`.

* Type récupéré lors de l'accès par colonne d'une dataframe : si df est un dataframe avec 'A' parmi ses colonnes :

- `df.loc[:,['A']]` est un dataframe (avec une seule colonne).
- `df.loc[:,'A']` est une series, comme `df['A']`.

Annexe

Dans Scikit les algorithmes de Machine Learning sont exposés via des objets appelés *"estimator"*.

Exemple pour une régression linéaire:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Pour tous les *estimators*:

- model.fit() : remplit le modèle avec des données d'entrainement. Pour un apprentissage supervisé, la méthode accepte 2 arguments: les données X et les labels y (i.e. model.fit(X, y)). Pour une apprentissage non supervisé, la méthode ne prend qu'un seul arguement, les données X (i.e. model.fit(X)).

Pour les *estimators* en apprentissage supervisé:

- model.predict() : prédire le label d'un ensemble de features à partir d'un modèle entrainé. La méthode accepte un argument, les nouvelles données X_new (i.e. model.predict(X_new) et retourne les labels prédits pour chaque objet du tableau.
- model.predict_proba() : Pour les problèmes de classification, certains *estimators* fournissent cette méthode qui retourne la probabilité qu'une nouvelle observation possède chaque label. La label qui la plus forte probabilité est retourné par model.predict().
- model.score() : Pour les problèmes de régession ou de classification, les *estimators* implémentent une méthode de score. Cette dernière permet d'indiquer si le fit est bon ou pas. Le score peut varier entre 0 et 1.