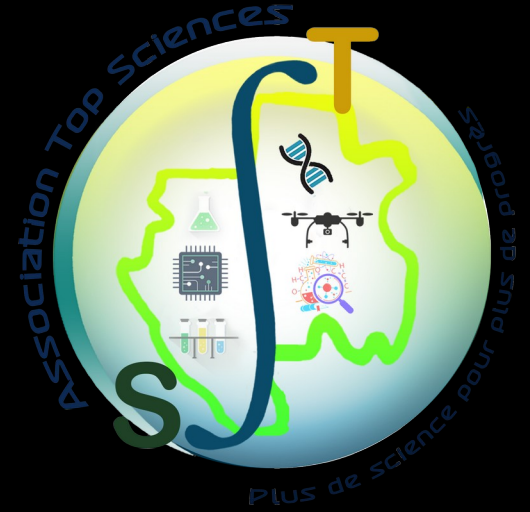


# Linear Regression

By Dr. Nzamba Bignoumba



$$y = -ax + b$$

Average training duration: 4 hours 00 minute

# Outline

Machine learning overview	→	15 min
Linear regression: theory	→	45 min
Linear regression: use case	→	01.30 h
Model deployment	→	01.30 h

# Machine learning overview

**IA**

Que pouvons nous faire  
avec ?

Natural Language Processing  
Computer Vision  
Signal Processing  
⋮

# Machine learning overview

Synthèse du contenu d'un ou plusieurs documents  
Traduction d'une langue à une autre  
Génération du code  
⋮

OpenAI-ChatGPT  
DeepSeek  
GitHub Copilot  
Cursor | Codex

Génération de contenu visuel  
Classification d'images médicales  
Classification d'images agricoles  
Détection d'objet  
⋮

Midjourney  
Canva  
Aidoc  
IA Agri  
AgriHyphen AI

Natural Language Processing  
Computer Vision  
Signal Processing  
⋮

Prévisions météorologiques  
Prévisions de maladies et de mortalité  
Prévisions boursières  
Prévisions de consommation d'électricité  
Détection d'anomalies (cybersécurité)  
⋮

AWS SageMaker - DeepAR  
Nixtla-TimeGPT  
Meta-Prophet  
Zindi Africa  
Amini

# Machine learning overview

IA

Que pouvons nous faire  
avec ?



Natural Language  
Processing Computer Vision  
Signal Processing  
⋮

Avec quels types  
d'algorithmes?



Machine Learning  
Deep Learning

# Machine learning overview

Support Vector  
Machines

Logistic  
Regression

K-NN

K-Means

Decision Trees

Gradient Boosting Machines

Random Forest

Linear  
Regression

Principal Component  
Analysis

Machine Learning

...

Deep Learning

Generative Adversarial  
Neural Network

Varational  
Autoencoder

Feed Forward  
Neural Network

State Space Model

Word  
Embeddings

Autoencoder

Graph Neural Network

Neural Ordinary  
Differential Equations

Diffusion Model

Recurrent  
Neural Network

Normalizing Flows

Transformer

Neural Radiance Field

...

# Machine learning overview

IA

Que pouvons nous faire  
avec ?



Natural Language  
Processing Computer Vision  
Signal Processing  
⋮

Avec quels types  
d'algorithmes?

Machine Learning

Deep Learning



Comment  
fonctionnent-ils ?

Supervised  
Non-Supervised  
Self-Supervised  
⋮



# Machine learning overview

Nous disposons des données  $\mathbf{X}$ s et des des cibles correspondantes  $\mathbf{y}$ s. Le modèle utilisera  $\mathbf{X}$  pour prédire une valeur  $\hat{\mathbf{y}}$ ,  $\text{modèle}(\mathbf{X})=\hat{\mathbf{y}}$ .  $\hat{\mathbf{y}}$  sera ensuite comparée à  $\mathbf{y}$  pour calculer l'erreur  $\mathbf{e}$  faite par le modèle,  $|\mathbf{y}-\hat{\mathbf{y}}|=\mathbf{e}$ . Cette erreur servira à **ajuster les paramètres** du modèle afin qu'il puisse prédire une valeur  $\hat{\mathbf{y}}$  très proche de  $\mathbf{y}$ .

Observation

$\mathbf{x}$  →  $\mathbf{y}$

Cible  
(target)

$\mathbf{e}$

Erreur que le modèle va  
chercher à minimiser.

Model

$\hat{\mathbf{y}}$

Prédiction



Supervised  
Non-Supervised  
Self-Supervised  
⋮



# Machine learning overview

Nous ne disposons que des données **Xs**. Aucune cible **y** correspondante n'est disponible. Le modèle exploitera les **similarités** des données et leur **cooccurrence** pour effectuer la tâche qui lui est assignée. Par exemple, la tâche de clustering, qui consiste à regrouper des données présentant des caractéristiques similaires (patterns).

Observation

**x**

~~**y**~~

Inexistante

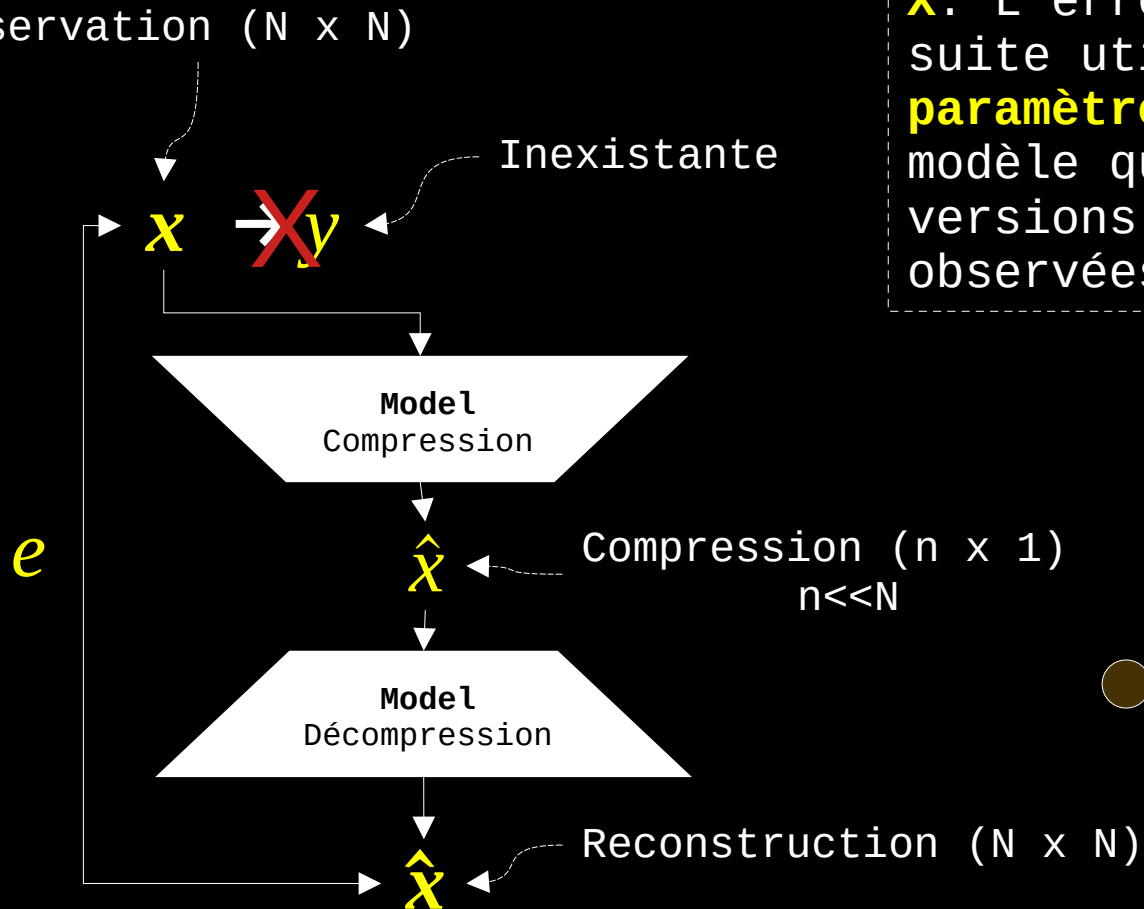
Model

Clustering

Supervised  
Non-Supervised  
Self-Supervised  
⋮

Machine learning  
overview

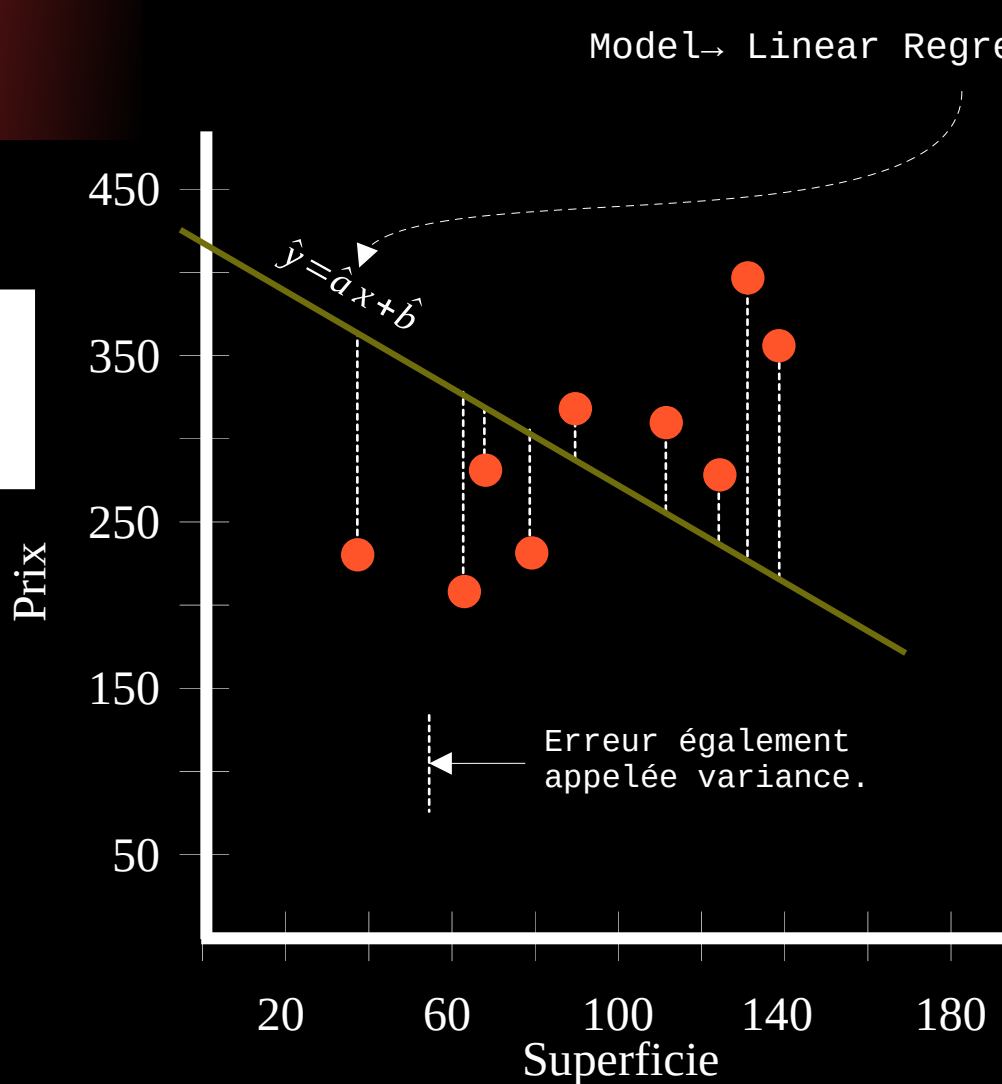
Nous ne disposons que des données **Xs**. Le modèle prend une observation **X** en entrée, **model(X)= $\hat{X}$** , et compare sa prédiction  **$\hat{X}$**  à cette même observation **X**. L'erreur  **$|X - \hat{X}| = e$** , sera par la suite utilisée pour **ajuster les paramètres** du modèle. Exemple : un modèle qui apprend à créer des versions compressée des données observées.



Supervised  
Non-Supervised  
Self-Supervised  
⋮

## Theory

Graphique des prix (en XAF) de location des maisons en fonction de leur superficie (en m<sup>2</sup>).



L'entraînement consiste à trouver les valeurs  $\hat{a}$  et  $\hat{b}$  qui minimisent au maximum l'erreur ( $E$ ).

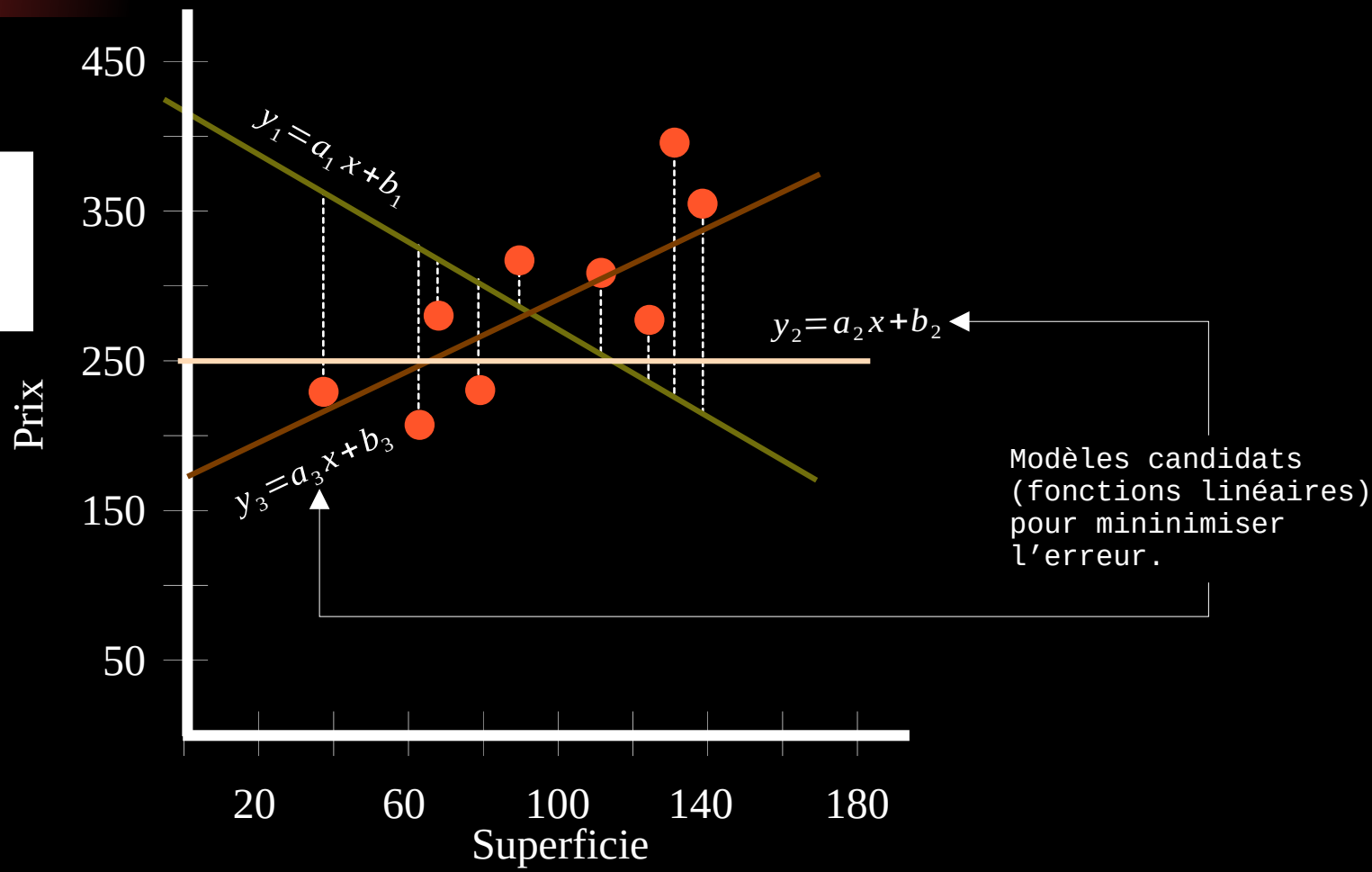
$$E = | \quad | + | \quad | + | \quad | + \dots |$$

$$E = (e_1 + e_2 + \dots + e_9) / I$$

$I=9$ , est le nombre de points.

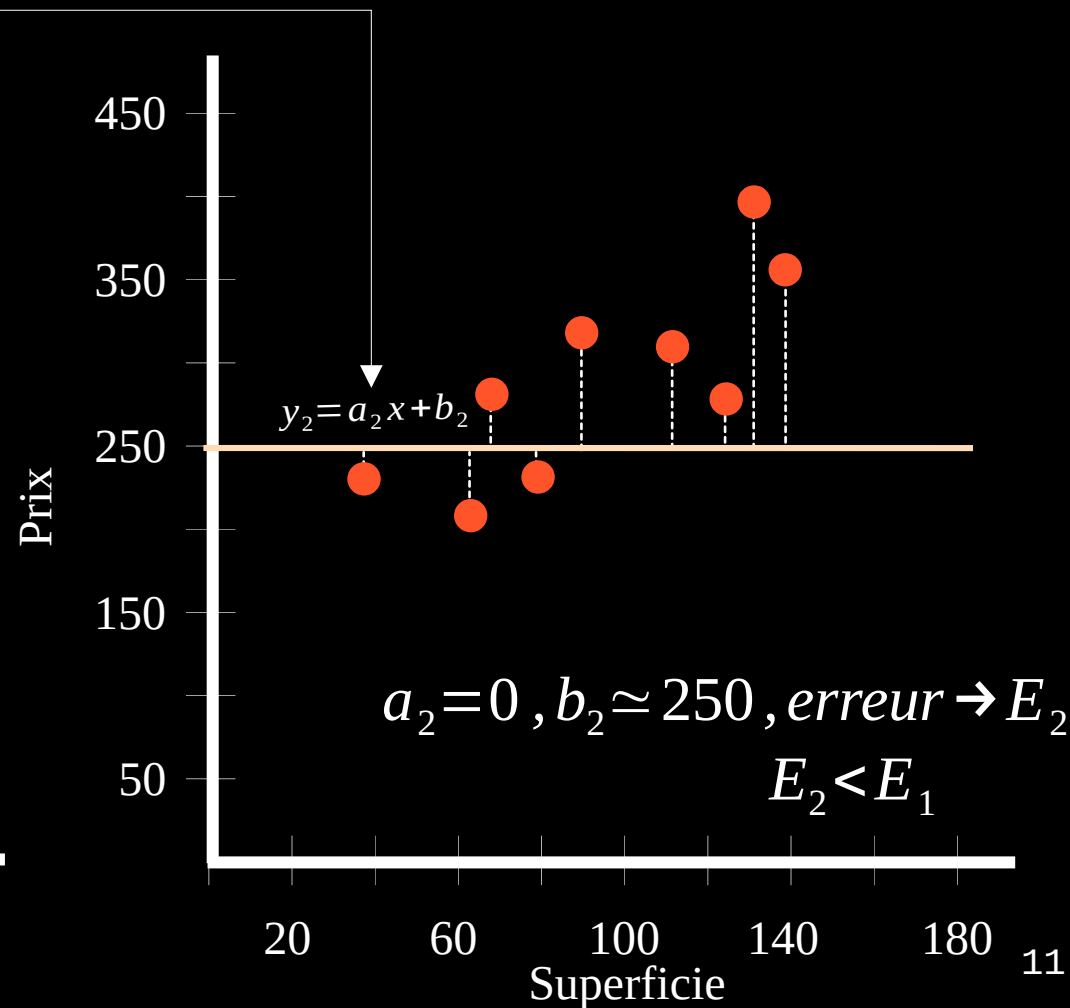
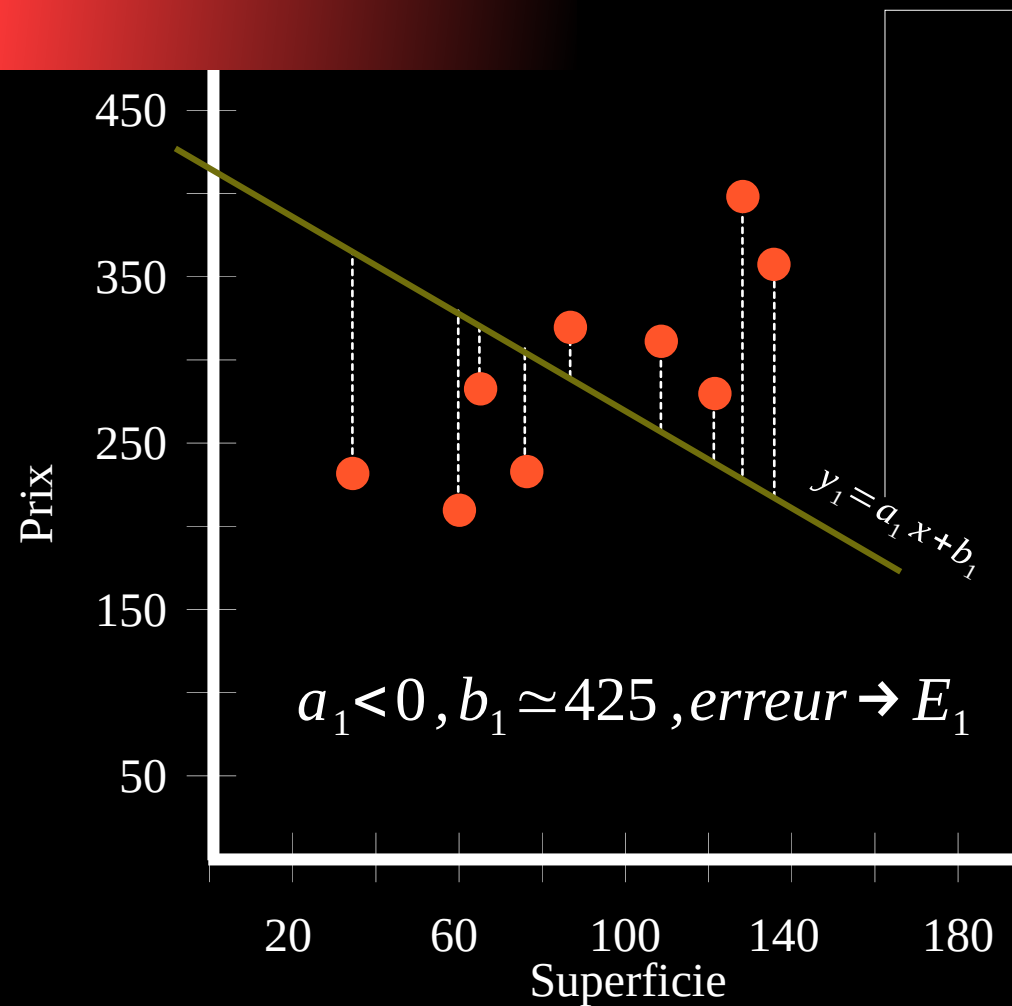
## Theory

Graphique des prix  
(en XAF) de location  
des maisons en  
fonction de leur  
superficie (en m<sup>2</sup>).

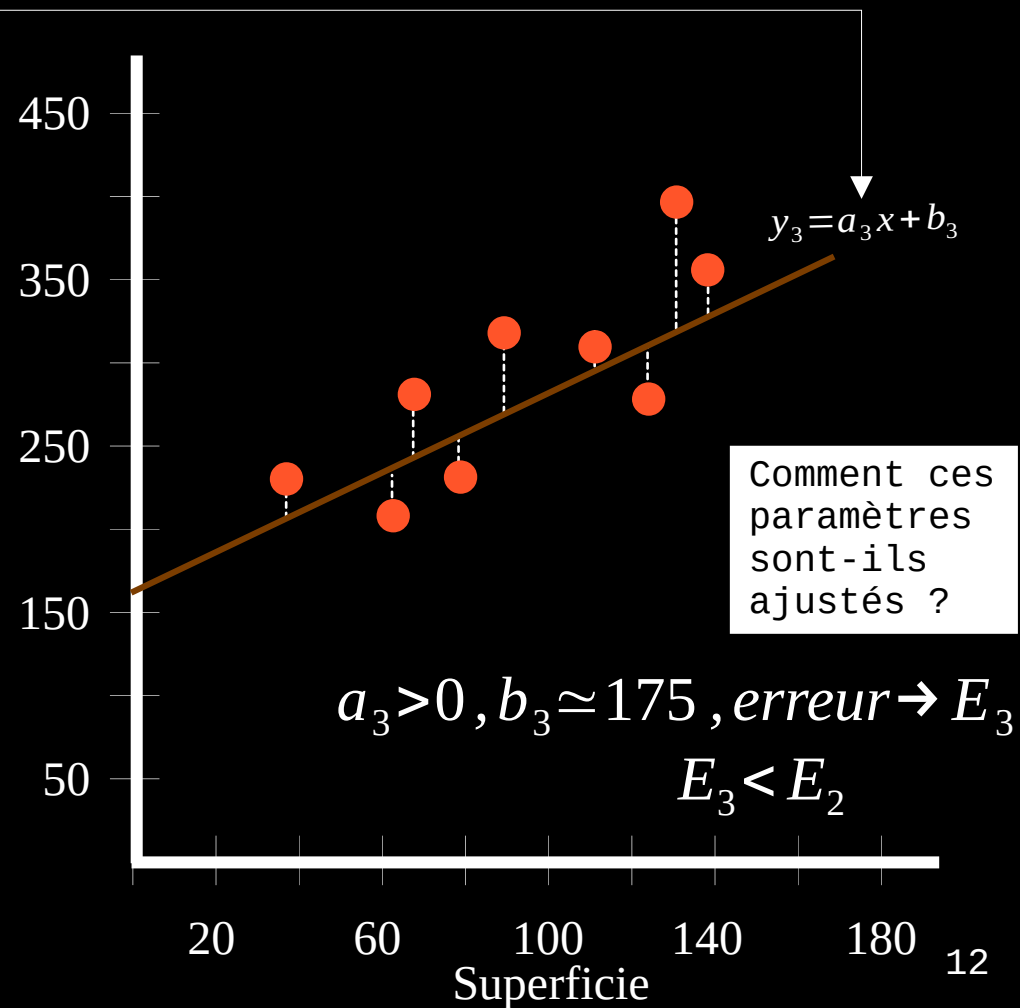
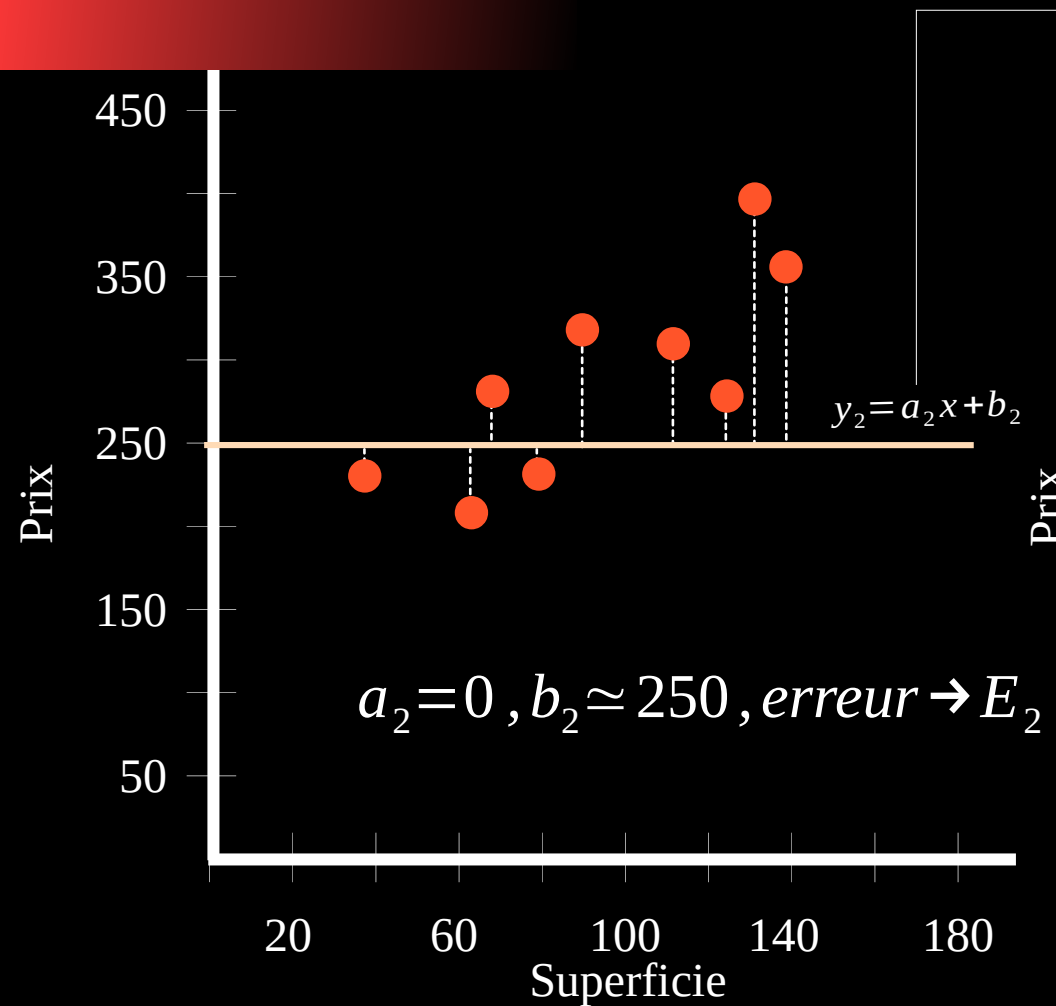


## Theory

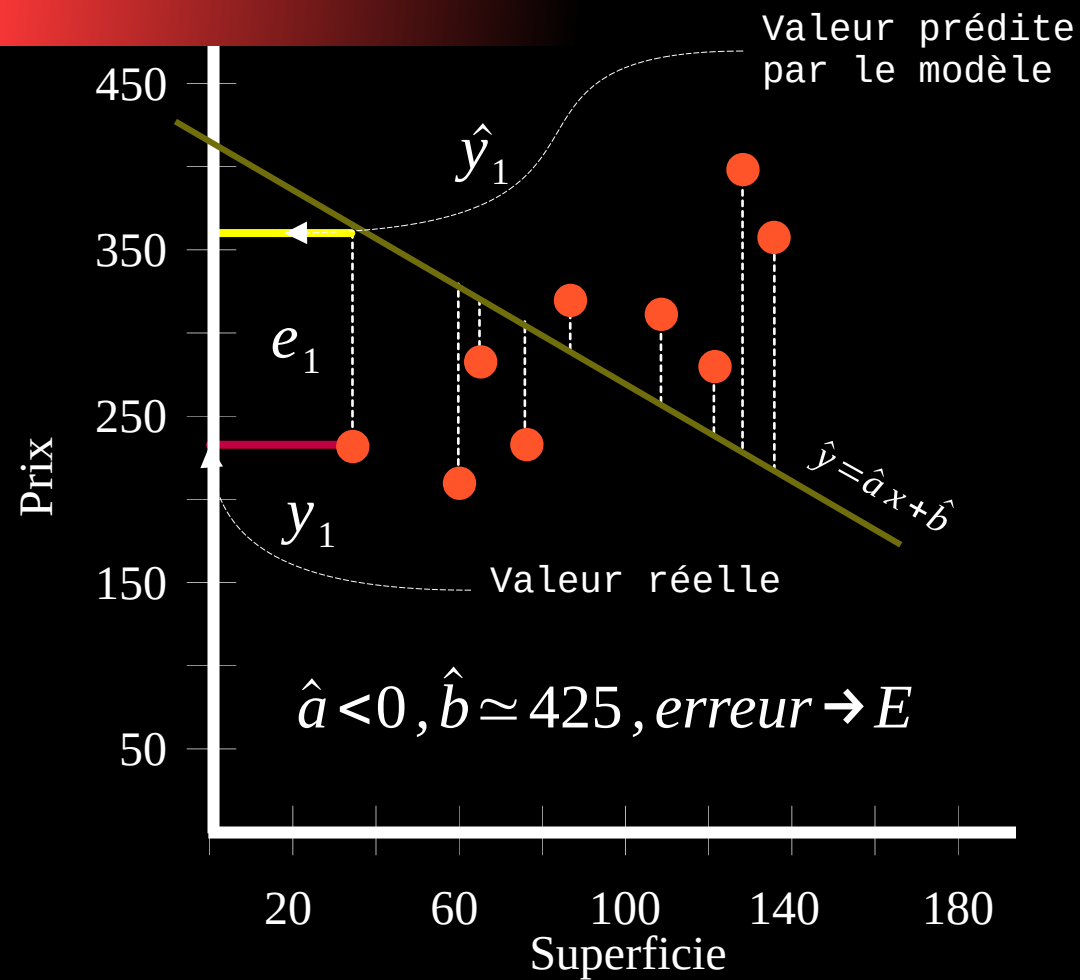
Amélioration du modèle après n entraînements.



## Theory

Amélioration du modèle après  $n+k$  ( $k>0$ ) entraînements.

## Theory



$$e_1 = (y_1 - \hat{y}_1)^2 = [y_1 - (\hat{a}x_1 + \hat{b})]^2$$

$$E = (e_1 + e_2 + \dots + e_9) / I = \frac{1}{I} \sum_{i=1}^I e_i$$

E est appelée **Mean Squared Error (MSE)** et I=9, est le nombre de points

$$E = \frac{1}{I} \sum_{i=1}^I [y_i - (\hat{a}x_i + \hat{b})]^2$$

Comment minimiser l'erreur E?

En trouvant les valeurs optimales de  $\hat{a}$  et  $\hat{b}$  qui minimise cette erreur.

Dérivons E pour trouver les paramètres optimaux.

## Theory

$$E = \frac{1}{I} \sum_{i=1}^I [y_i - (\hat{a} x_i + \hat{b})]^2$$

$$[(u - v)^2]' = 2(u - v)(u' - v'), u = y_i \text{ and } v = (\hat{a} x_i + \hat{b})$$

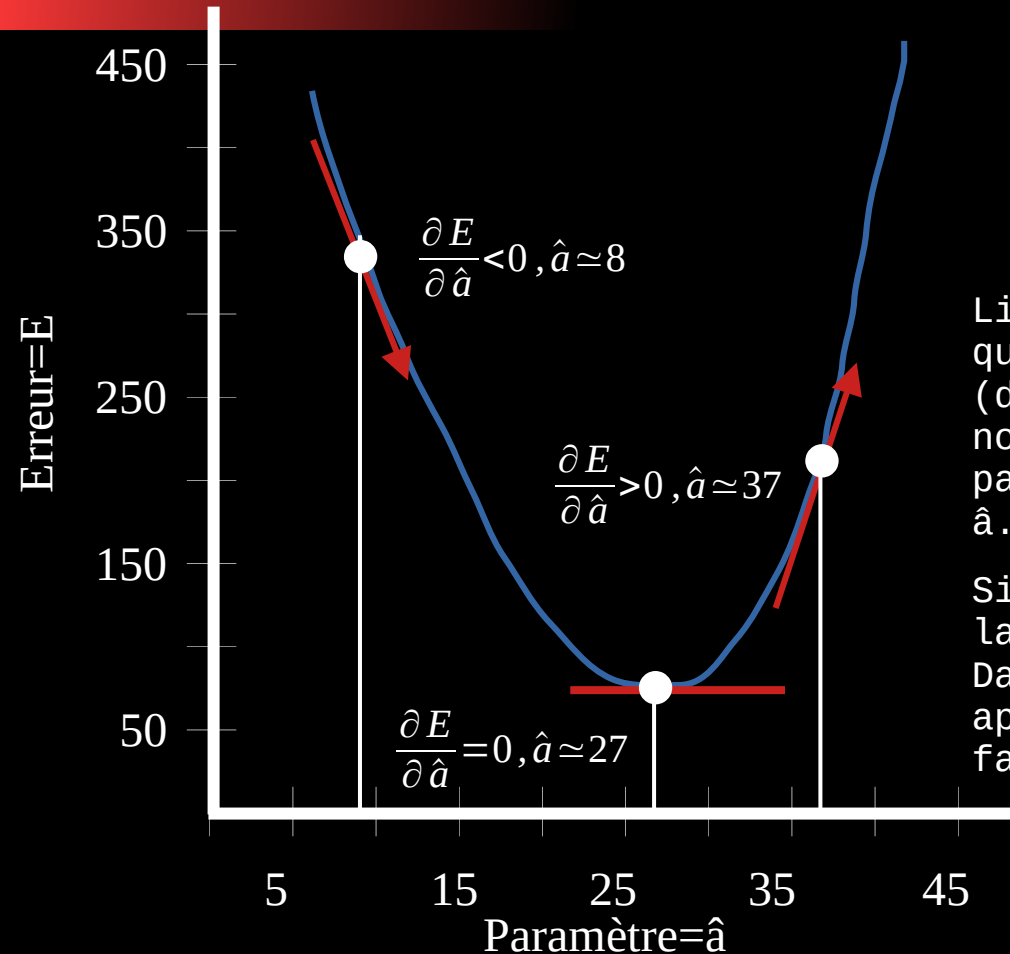
$$\frac{\partial E}{\partial \hat{a}} = \frac{1}{I} \sum_{i=1}^I 2[y_i - (\hat{a} x_i + \hat{b})](-x_i) = -\frac{2}{I} \sum_{i=1}^I [y_i - (\hat{a} x_i + \hat{b})](x_i)$$

$$\frac{\partial E}{\partial \hat{b}} = \frac{1}{I} \sum_{i=1}^I 2[y_i - (\hat{a} x_i + \hat{b})](-1) = -\frac{2}{I} \sum_{i=1}^I [y_i - (\hat{a} x_i + \hat{b})]$$



## Theory

## Rappel sur les dérivés des fonctions



Supposons que nous voulons calculer la dérivée de E au point  $\hat{a}=x$ , formellement on aura:

$$E'(\hat{a}=x) = \frac{\partial E}{\partial \hat{a}}$$

Littéralement, la fonction ci-dessus signifie: quelle est le taux de changement de E (diminution, augmentation, stagnation), lorsque nous sommes au point  $\hat{a}$ . Ce taux est représenté par un vecteur tangent à la courbe de E au point  $\hat{a}$ .

Si nous voulons minimiser E, nous devons trouver la valeur  $\hat{a}$  qui conduit à cette minimisation. Dans notre exemple, cette valeur est approximativement égale à 27. Ceci doit aussi se faire pour  $\hat{b}$ .

Calculons la dérivée de E en fonction de  $\hat{a}$  et  $\hat{b}$ .

## Theory

## Rappel algèbre linéaire

Addition de  
vecteurs

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_k + b_k \end{bmatrix}$$

Exemple  $\rightarrow \begin{bmatrix} 2.5 \\ 18.0 \\ \vdots \\ 7.5 \end{bmatrix} + \begin{bmatrix} 3.9 \\ 8.1 \\ \vdots \\ 4.2 \end{bmatrix} = \begin{bmatrix} 6.4 \\ 26.1 \\ \vdots \\ 11.7 \end{bmatrix}$

Produit d'un  
nombre réel avec  
un vecteur

$$X \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} Xa_1 \\ Xa_2 \\ \vdots \\ Xa_k \end{bmatrix}$$

Exemple  $\rightarrow 2 \begin{bmatrix} 3.9 \\ 8.1 \\ \vdots \\ 4.2 \end{bmatrix} = \begin{bmatrix} 7.8 \\ 16.2 \\ \vdots \\ 8.4 \end{bmatrix}$

## Theory

## Rappel algèbre linéaire

Pour multiplier deux matrices, le nombre de colonnes de la première matrice doit être égal au nombre de lignes de la seconde.

Produit de matrices

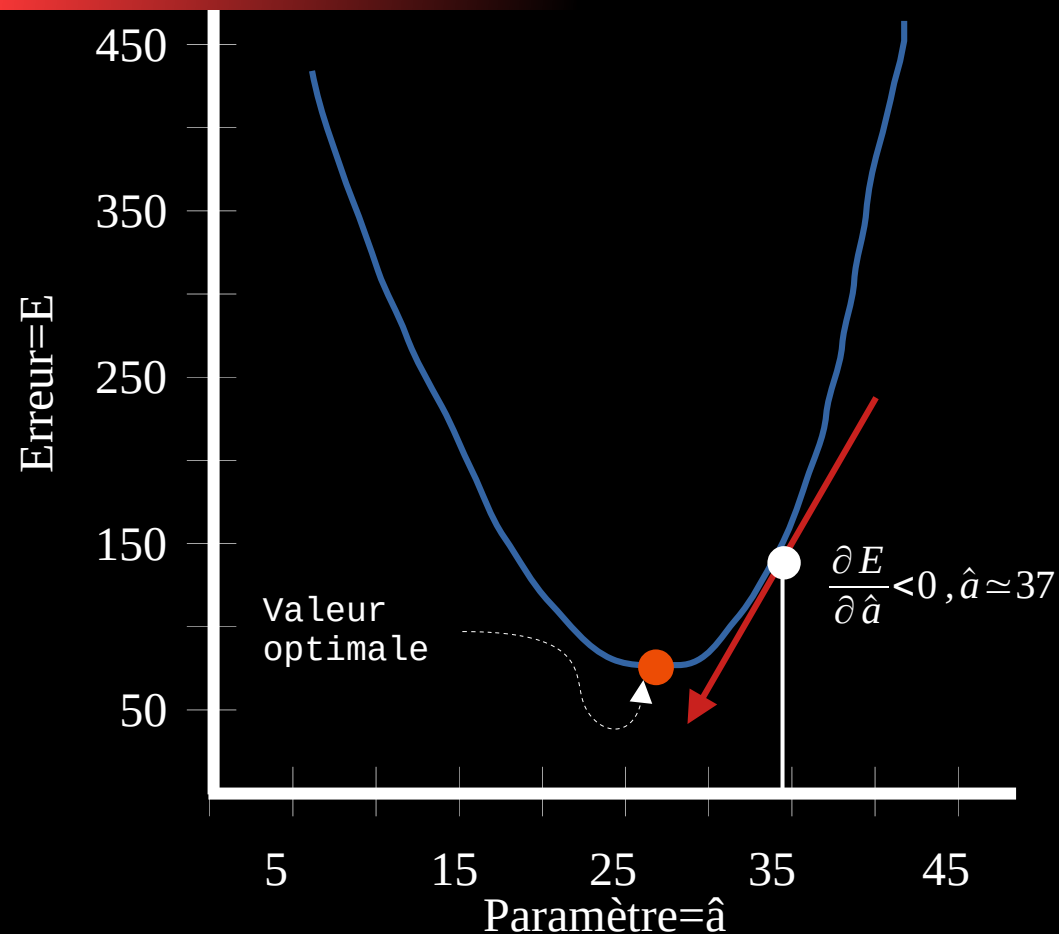
$$\begin{matrix} 2 \times 2 & 2 \times 3 & & 2 \times 3 \\ \begin{bmatrix} a_1^1 & a_2^1 \\ a_1^2 & a_2^2 \end{bmatrix} & \begin{bmatrix} b_1^1 & b_2^1 & b_3^1 \\ b_1^2 & b_2^2 & b_3^2 \end{bmatrix} & = & \begin{bmatrix} a_1^1 b_1^1 + a_2^1 b_1^2 & a_1^1 b_2^1 + a_2^1 b_2^2 & a_1^1 b_3^1 + a_2^1 b_3^2 \\ a_1^2 b_1^1 + a_2^2 b_1^2 & a_1^2 b_2^1 + a_2^2 b_2^2 & a_1^2 b_3^1 + a_2^2 b_3^2 \end{bmatrix}
 \end{matrix}$$

## Exemple

$$\begin{bmatrix} 2,1 & 4,3 \\ 1,7 & 7,0 \end{bmatrix} \begin{bmatrix} 2,5 & 10,0 & 13,2 \\ 5,4 & 4,4 & 6,5 \end{bmatrix} = \begin{bmatrix} 2,1*2,5+4,3*5,4 & 2,1*10,0+4,3*4,4 & 2,1*13,2+4,3*6,5 \\ 1,7*2,5+7,0*5,4 & 1,7*10,0+7,0*4,4 & 1,7*13,2+7,0*6,5 \end{bmatrix}$$

$$\begin{bmatrix} 2,1 & 4,3 \\ 1,7 & 7,0 \end{bmatrix} \begin{bmatrix} 2,5 & 10,0 & 13,2 \\ 5,4 & 4,4 & 6,5 \end{bmatrix} = \begin{bmatrix} 28,47 & 39,92 & 55,67 \\ 42,05 & 47,8 & 67,94 \end{bmatrix}$$

## Theory



La nouvelle valeur de  $\hat{a}$  sera:

$$\hat{a} = \hat{a} - \eta \frac{\partial E}{\partial \hat{a}}$$

$\eta > 0$  Est appelée **learning rate**.  
Elle est souvent égale à 0.001

Ce processus d'ajustement appelé **descente de gradient** est répété  $n$  fois jusqu'à ce que l'on trouve la valeur optimale de  $\hat{a}$  (ainsi que  $\hat{b}$ ) qui minimise  $E$ .

Que se passe-t'il si nous avons plusieurs valeurs observées ?

## Theory

$$\hat{y} = \hat{a}_1 x_1 + \hat{a}_2 x_2 + \dots + \hat{a}_k x_k + \hat{b} = [x_1, x_2, \dots, x_k] \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_k \end{bmatrix} + \hat{b}$$

$$\hat{y} = X\theta + \hat{b} \mid X = [x_1, x_2, \dots, x_k], \theta = \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_k \end{bmatrix}$$

$$E = \frac{1}{I} \sum_{i=1}^I [y_i - (X_i \theta + \hat{b})]^2$$

$$X = [x_1, x_2, \dots, x_k] \rightarrow X^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

$$\nabla_{\theta} E(\theta) = \frac{1}{I} \sum_{i=1}^I 2[y_i - (X_i \theta + \hat{b})](-X_i)^T = -\frac{2}{I} \sum_{i=1}^I [y_i - (X_i \theta + \hat{b})](X_i)^T$$

## Theory

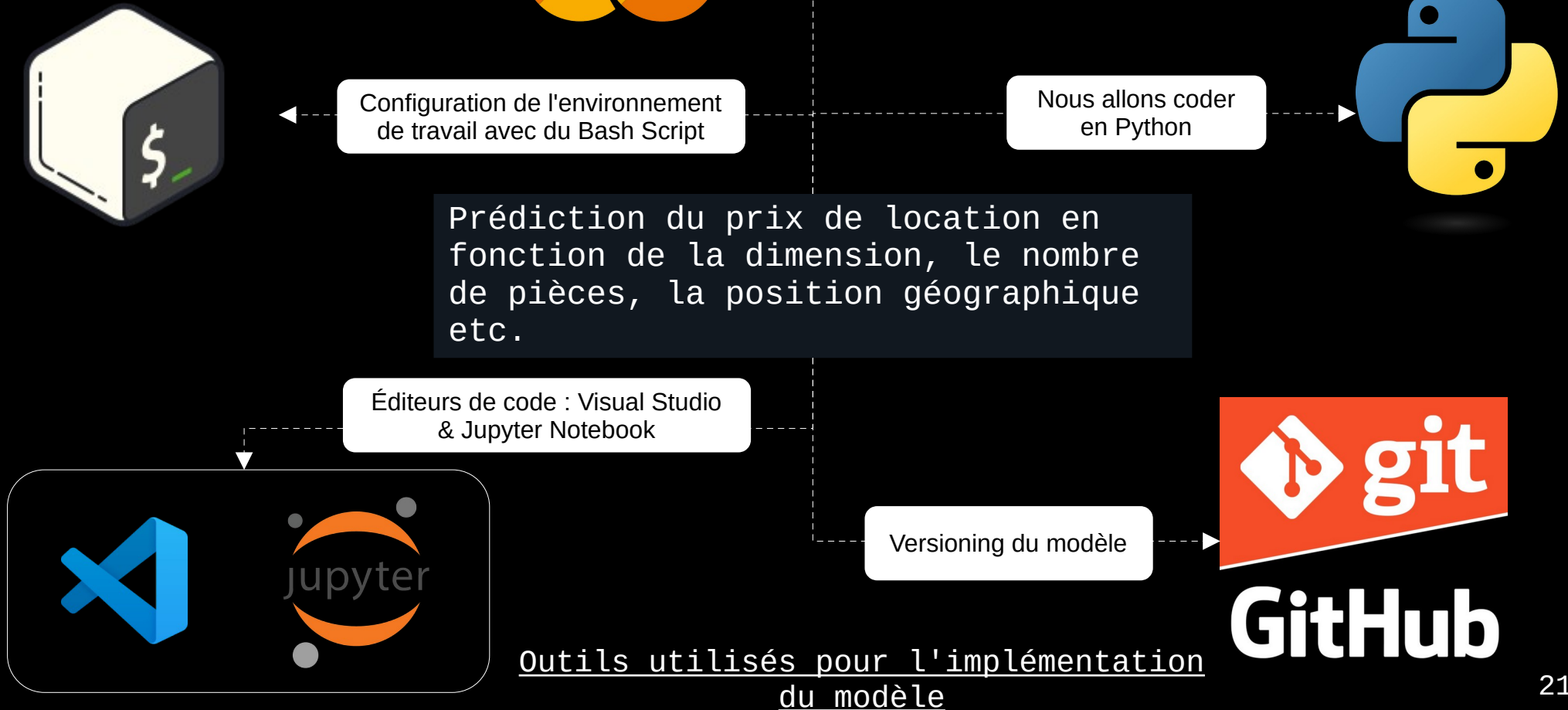
$$X = [x_1, x_2, \dots, x_k] \rightarrow X^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

$$\alpha_i = [y_i - (X_i \theta + \hat{b})]$$

$$\nabla_{\theta} E(\theta) = \frac{-2}{I} \sum_{i=1}^I [y_i - (X_i \theta + \hat{b})] (X_i)^T = \frac{-2}{I} \sum_{i=1}^I \alpha_i \begin{bmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_k^i \end{bmatrix}$$

$$\theta = \theta - \ni \nabla_{\theta} E(\theta) = \theta - \ni \frac{-2}{I} \sum_{i=1}^I \alpha_i \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_k \end{bmatrix} - \ni \frac{-2}{I} \sum_{i=1}^I \alpha_i \begin{bmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_k^i \end{bmatrix}_{20}$$

## Use Case



- *Python est un langage de programmation codé en C;*
- *Langage impératif et interprété;*
- *Langage le plus utilisé en data science;*
- *Inclut diverses bibliothèques telles que Pandas pour la manipulation de données structurées, Numpy pour les calculs mathématiques, etc.*



## Use Case

```

observations=[ 'pieds_carres','nombre_chambres','nombre_etages','distance_centre_ville','annee_construction','taille_garage','score_localisation','avec_piscine']
# observations=[ 'pieds_carres','nombre_etages','distance_centre_ville','annee_construction']

X=data[observations].values#Observations

print('Exemple des observations/features avant la standardization.')
print(X[0])

scaler = StandardScaler()
X_ = scaler.fit_transform(X)
print('\n')
print('Exemple des observations/features après la standardization.')
print(X_[0])

y=data[['prix']].values#Target (cible)
#2% des données (soit 100 échantillons/samples) seront utilisées pour le test.
#random_state permet de reproduire la distribution des données d'entraînement et de test.
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1234)

X_train=scaler.fit_transform(X_train)#Standardiser les données d'entraînement.

```

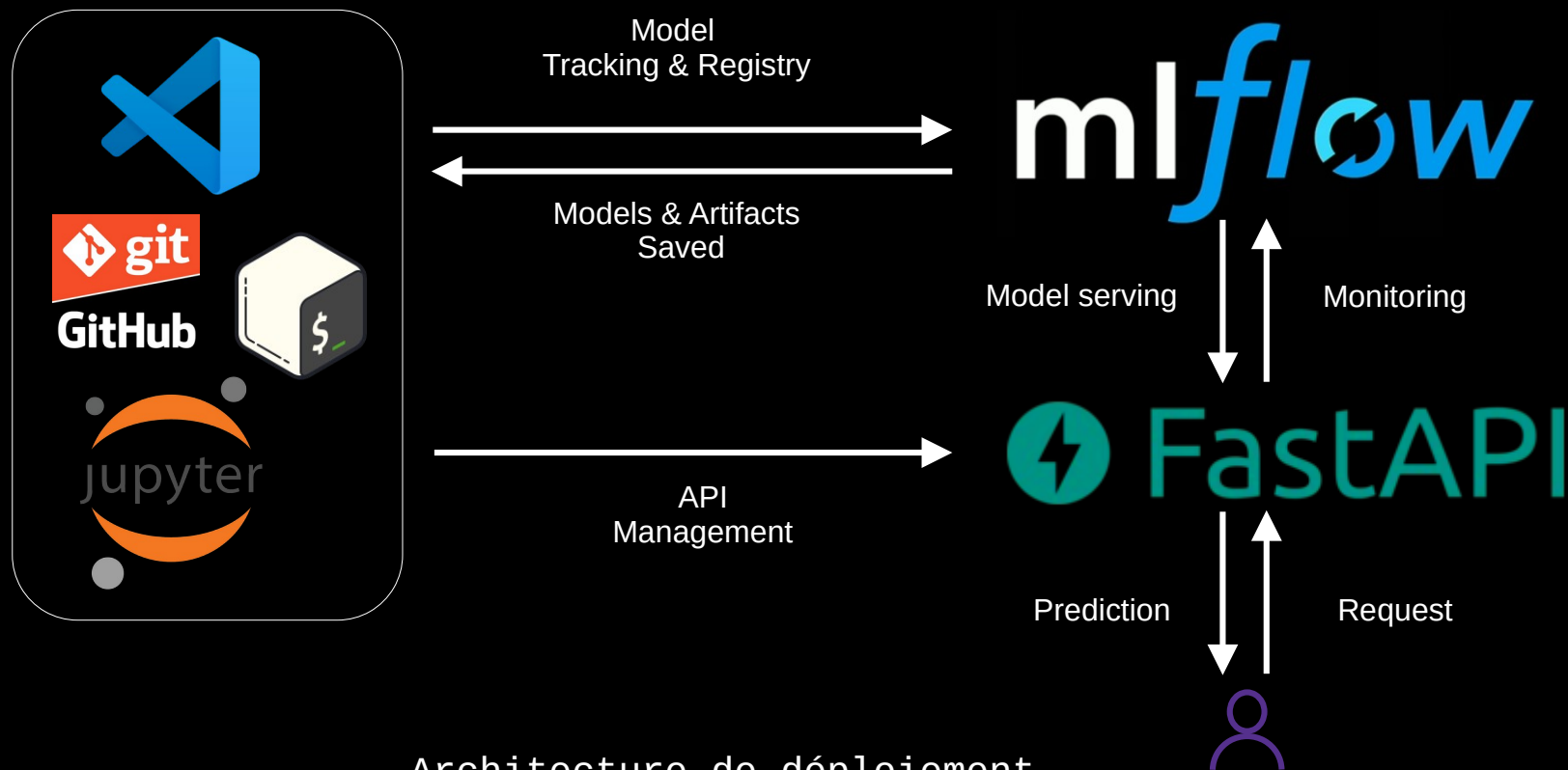
*Codons*

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import mlflow
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from datetime import time
from sklearn.preprocessing import StandardScaler
import random

```

## Model deployment



Architecture de déploiement

